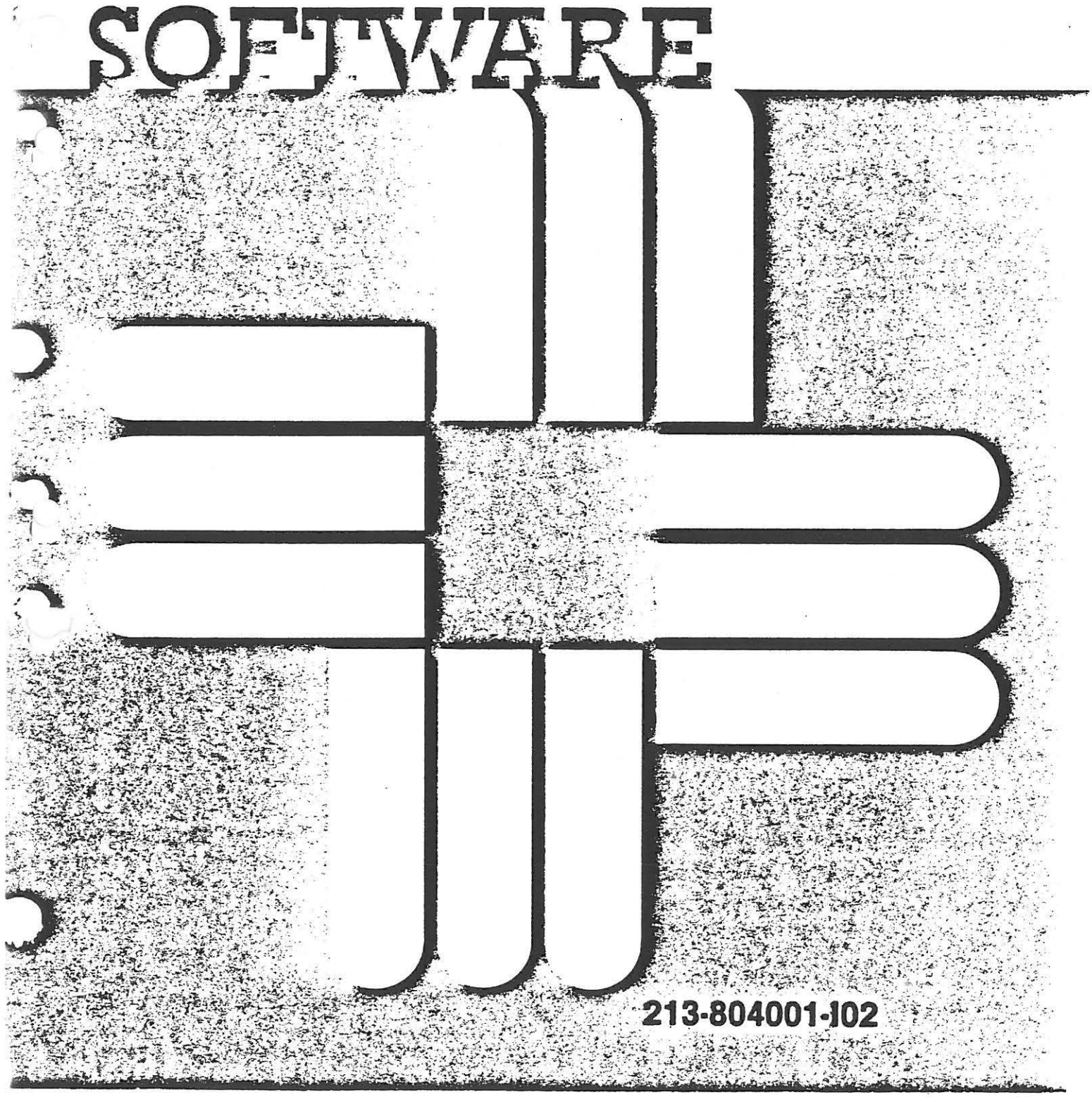
MODCOMP
MODULAR COMPUTER SYSTEMS, INC.

MAX IV System Guide

General Operating System

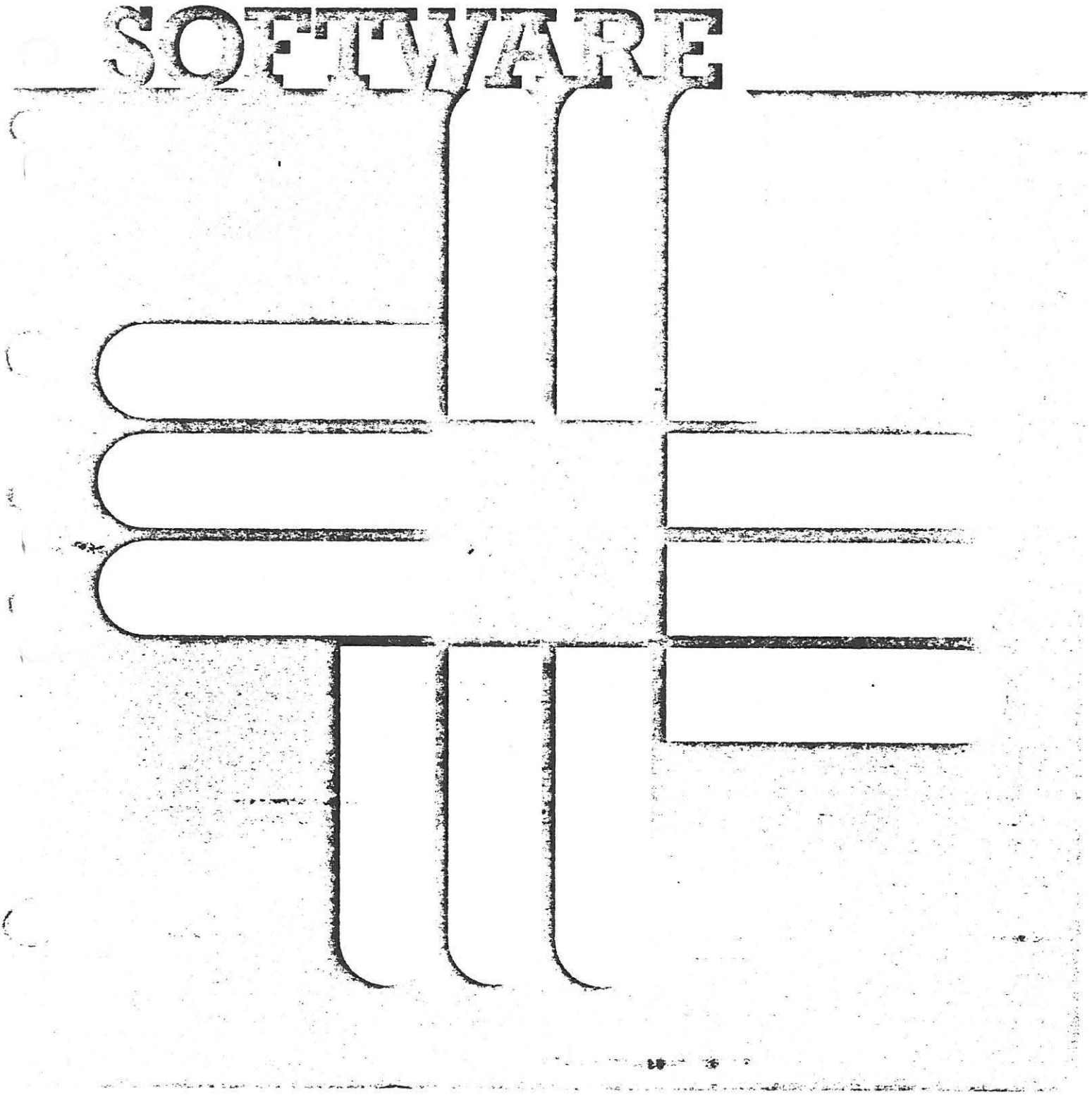


213-804001-102

MODCOMP
MODULAR COMPUTER SYSTEMS, INC.

MAX IV System Guide

General Operating System



MANUAL HISTORY

Manual Order Number: 213-804001-I02

Title: MAX IV System Guide, General Operating System

Software Part Number: 610304-000I.2

Model Number: 8240I

Revision Level	Date Issued	Description
H00	05/83	Initial issue (H.0). This manual replaces Chapters 2, 3, 4, 6, 10, and 11 of manual order number 210-610304-000 and Chapters 7, 8, 9, and 11 of manual order number 220-610304-000.
H01	12/83	Revision (H.1).
H02	06/84	Revision (H.2).
I00	09/84	Reissue (I.0). Included additional information for Module Loaders. Several new directives were added, primarily for the new MAGIC product.
I01	08/85	Revision (I.1). Technical changes affect the following subjects: Moving optional features of the Operating System out of MAP 0 using MP.TSK; date/time stamp option of the TMP MSG and BROADCAST commands; new output format for the TMP NUM command; U$LOA routine for calls to the Module Loader; no bias for OC directives that are moved out of MAP 0.
I02	01/86	Revision (I.2). Documentation for a new TMP overlay service routine, OSWBA, has been added to Chapter 5.

Contents subject to change without notice.

Copyright © 1986, by Modular Computer Systems, Inc.
All Rights Reserved.
Printed in the United States of America.

PROPRIETARY INFORMATION

THIS MANUAL CONTAINS CONFIDENTIAL PROPRIETARY INFORMATION PROTECTED BY SOFTWARE PROPERTY RIGHTS AND IS MADE AVAILABLE UPON THE CONDITION THAT THE SOFTWARE CONTAINED HEREIN WILL BE HELD IN ABSOLUTE CONFIDENCE AND MAY NOT BE DISCLOSED IN WHOLE OR IN PART TO OTHERS WITHOUT THE PRIOR WRITTEN PERMISSION OF MODULAR COMPUTER SYSTEMS, INC.

FOR GOVERNMENT USE THE FOLLOWING SHALL APPLY:

RESTRICTED RIGHTS LEGEND

USE, DUPLICATION OR DISCLOSURE BY THE GOVERNMENT IS SUBJECT TO RESTRICTIONS AS SET FORTH IN PARAGRAPH (b) (3) (B) OF THE RIGHTS IN TECHNICAL DATA AND COMPUTER SOFTWARE CLAUSE IN DAR7-104.9(a) (OR SUCH EQUIVALENT FPR, NASPR, AND/OR DOEPR CLAUSE AS MAY BE APPLICABLE).

MODULAR COMPUTER SYSTEMS, INC.
1650 WEST McNAB ROAD
FORT LAUDERDALE, FL 33309

PREFACE

Audience: The information in this manual is directed to the applications programmer or developer with a good knowledge of MODCOMP Assembler language and familiarity with MODCOMP terminology and equipment.

Subject: This manual contains information about the MAX IV General Operating System.

Those users familiar with previous issues of MAX IV Operating Systems manuals will notice that the information in this manual appeared in either:

MAX IV General Operating System Reference Manual
Chapters 1, 2, 3, 4, 6, 10, and 11

MAX IV General Operating System Technical Manual
Chapters 2, 7, 8, 9, and 11

Products Supported: MAX IV Software System, Model 8240I.

Special Symbols and Notations: A revision bar (|) located in the margin of the page in the text indicates a change to the manual. This change generally represents a technical change to the product due to product revision. A revision bar is also entered in the Table of Contents to flag the general location of changes in the text.

Related Publications: The reader is referred to the manuals listed below for additional information.

Manual Order Number	Manual Title
211-804011-REV	TASK/OVERLAY CATALOGER Programmer's Reference Manual
209-804001-REV	MAX IV FILE MANAGER File Management Manual
210-804002-REV	MAX IV FORTRAN IV Language Reference Manual
213-804002-REV	MAX IV SYSGEN System Guide Manual
213-804003-REV	MAX IV EXECUTIVE (REX) SERVICES System Guide Manual
213-804004-REV	MAX IV OPERATOR COMMUNICATIONS System Guide Manual

213-804005-REV	MAX IV BASIC INPUT/OUTPUT SYSTEM System Guide Manual
211-804002-REV	MAX IV/MAX 32 NONRESIDENT JOB CONTROL AND BATCH FACILITIES, Programmer's Reference Manual
211-804009-REV	MAX IV LINK EDITOR (EDI) Programmer's Reference Manual

When ordering manuals, use the Manual Order Number listed in this Preface. The lastest revision (REV) will be shipped.

MAX IV REVISION I.2 SUMMARY OF CHANGES

TMP

A new overlay service routine, OSWBA, is available. Overlays can now use this routine or the OSWRI routine for performing output to terminals. However, only OSWBA allows the specification of a buffer address from where the message is to be printed.

MAX IV REVISION I.1 SUMMARY OF CHANGES

NONMAP Ø CODE

If any optional features of the operating system are to be moved out of MAP Ø, only two packages can be moved per MP.TSK. In addition, the File Manager and FORTRAN Runtime packages cannot be selected together as two options of the same MP.TSK.

TMP TASK VARIABLES

In the description of the WPT task variable (Write Prompt Timer Flag/Value), the documentation was changed to clarify that one tick is equal to five milliseconds (.ØØ5 second).

TMP DIRECTIVES

Messages sent through the BROADCAST and MSG commands are preceded to a date/time stamp if the 'DTS' option was specified at SYSGEN.

The receiver of a message sent through the MSG command can be identified by either the terminal (port) number or User ID.

The documentation for the NUM Command was rewritten. The format of the output generated by this command has changed, although the same information is reported and no changes were made to the command's functionality or usage.

MODULE LOADER

Section 8.9 was added to describe the U$LOA routine that is supplied with the operating system. This routine is used for calls to the Module Loader (ML$LOD).

OPERATOR COMMUNICATIONS

OC directives must not be biased for loading into MAP Ø. The documentation in Appendix J that referenced a bias of absolute #FE for OC directives was deleted.

DCS

With the release of Revision I.1, MAX IV now supports the Distributed Communications Subsystem (DCS). DCS is the first of a new generation of MODCOMP communications controllers. It consists of up to 16 intelligent line interface modules (LIM's) which are interfaced by a Block Multiplexed Controller (BMUX) to the standard MODCOMP I/O bus. Each LIM has a Motorola 68000 microprocessor and supports up to four full duplex communications ports. Under DCS, the communications handlers actually run on the LIM card. A BMUX handler runs in the CLASSIC that passes BIOS requests (command packets) to the LIM and passes the results (status packets) back to the user task. All packets are transferred in DMP mode. With the initial release of DCS, three LIM handlers are supported: async, bit sync (SDLC), and MAXNET communications. The MAXNET handler supports async, byte sync, and SDLC type MAXNET links. DCS is documented in the MAX IV Distributed Communications Handler, System Guide (manual order number 213-804009-REV).

MAX IV REVISION I.Ø SUMMARY OF CHANGES

FILE MANAGER REVISIONS

- The REX call macros (FMRMAC) were updated to allow specification of the "Bypass SYSGEN Default FDL(BSD)" option on all File Manager REX calls. Documentation includes the addition of macro usage examples.

- A new service option was added to the FINFORMATION REX Service that verifies whether the supplied IAM descriptor is the Security Package Master UID.

- The Descriptor macros (FMRMAC) were updated to correct the allowable value specifications of descriptors. The descriptors and their values are described. Documentation includes the addition of Job Control specification examples plus the addition of any differences between the macro language and the Job Control specifications.

- An overview of the FMSAVE utility has been included.

JOB CONTROL REVISIONS

The following enhancements have been included to operate under the Revision I.Ø of the MAX IV Operating System or Revision A.Ø of the MAX 32 Operating System.

PROCEDURE DEFAULTS

The % (percent sign) can be used to default to an earlier parameter.

COMMANDS

Commands can be in upper or lower case.

$IF STATEMENTS

A new construct to the $IF statements uses THEN and ELSE as well as PRODUCE and semicolon(;).

$EXE STATEMENTS

A double comma in $EXE statements designates a missing option.

CHANGES FOR FILE MANAGER

The new File Manager Directive $LFILE was added to list the contents of any File Manager directory, data or partition data file.

All File Manager directives now have an option (BS) to specify that the requested service is to be performed without using the default SYSGENed File Descriptor List (FDL).

The $FDx Directive has been modified to list the contents of a user's File Descriptor List.

All File Manager directives can contain a filename (or volume name for volume related services) in place of a FDL specification (FDx). Other descriptors can be specified after this filename. This allows a File Manager call without previously defining an FDL.

All File Manager directives can contain an embedded volume name within specified filename.

All File Manager errors list a text error message as well as the hexadecimal error code.

SYSGEN REVISIONS

MULTIBATCH SYSTEMS

A new option, ROLL-ON-HOLD, has been added to the TASKMASTER SYSGEN macro to change the ROUNDROBIN algorithm to benefit a multi-batch environment. New OC directives, TMP directives, and JM overlays have been added to benefit a multi-batch environment.

INSTALLATION CONSIDERATIONS

Changes to the SYSGEN for installation of MAX IV Revision I.Ø do not need to be made unless the user wants to take advantage of the two new options mentioned below or unless WORKBENCH is running.

- A WORKBENCH SYSGEN no longer needs a second WORKBENCH macro in the I/O section of the SYSGEN. One WORKBENCH macro in Part One of the SYSGEN is all that is required. Remove the WORKBENCH in Part Two of the SYSGEN, the Basic I/O Section, and reSYSGEN.

The following SYSGEN statement has been added in Revision I.Ø of the MAX IV Operating System.

- TOCLOCK

TOCLOCK

A new data structure is created by the SYSGEN which is used for communication purposes between module loader and TOC rev L.Ø. This communication provides the security which makes the online compression of load module files safe.

This SYSGEN macro creates four entries (by default) for use by TOC and the module loader. The default to four entries may be too large or too small for a system. As a provision to change the number of structure entries, a TOCLOCK macro can be added to the SYSGEN.

Systems performing several load module file updates simultaneously, or performing load module file updates while running shared or segmented programs may need more than four entries. Realtime

systems that never run batch programs and that are MAP Ø tight can remove all entries by specifying Ø entries on the TOCLOCK macro.

NOTE: TOCLOCK Ø saves MAP Ø space but removes security for online load module file manipulation.

The following SYSGEN statements were modified in Revision I.Ø of the MAX IV Operating System.

- TASKMA (TASKMASTER)
- COMDEV (COMDEVICE)

TASKMA (TASKMASTER)

The ROLL-ON-HOLD option in the TASKMASTER statement can be specified for multi-batch environments. Parameter 3 allows the user to roll out tasks that are in I/O HOLD, OC HOLD, or are suspended. This affects only those tasks that are cataloged in the Task/Overlay Cataloger as WORKBENCH or peculiar batch.

COMDEV (COMDEVICE)

Bit 15 (PLC) defines the communication device types.

REX SERVICES REVISIONS

The following REX Services have been modified.

- ASSIGN
- TASSIGN
- TINFORM
- INIRES

ASSIGN

The MAX IV Revision I.Ø REX ASSIGN (#A) was modified to include a MOVE option. When the option is requested, the current assignment and file position index are copied from a specified File Assign Table (FAT) entry into a second FAT entry. The second FAT entry is available for use as a logical file of the calling task.

TASSIGN

The MAX IV Revision I.Ø REX TASSIGN (#B) was modified for the standard device information request to optionally return a File Manager File Assign Table (FAT) entry name instead of a #FFFE for the device name.

TINFORM

The MAX IV Revision I.Ø REX TINFORM (#43) was modified to optionally return a File Assign Table (FAT) entry name when a FAT index is specified.

INRES (Renamed to PROCESSOR REX Service)

Prior to MAX IV Revision I.Ø the INIRES RES Service (#41) was called exclusively by TOC to initialize resident load module directories. In revision I.Ø the scope of this service has been expanded to include special purpose functions needed by several MODCOMP processors. It now has a new name, the PROCESSOR REX Service. It is called by the Task/Overlay Cataloger (TOC), the Dump Analyzer (ANL4), and by Job Control. This service should only be used by MODCOMP PROCESSORS.

The INIRES option of the PROCESSOR REX Service initializes either named directories or all load module file resident directories to either the loaded or empty state. The purpose of this service is for TOC to update a load module file's resident directory following an updating function such as a catalog or compress.

TOCLOCK (option of the PROCESSOR REX)

This is an interface between TOC and the module loader. It prevents conflicts between module load and online load module file updates.

ANLGET (option of the PROCESSOR REX)

This returns information from memory for the Dump Analyzer.

DEFMODE (option of the PROCESSOR REX)

This resets Job Control to its original mode so that it can run under both MAX IV and MAX 32.

DATE/TIME STAMP (option of the TIME REX)

The date/time stamp is a new feature added to the TIME REX Service that formats an 18 ASCII character string containing the current date and time and returns it to the calling task by a task provided buffer. This REX Service will by used by MAX IV and by all new processors to standardize the format of system date and time displays. Older processors will gradually be converted to use this routine instead of doing their own formatting of date and time.

- The format of system date will have the month displayed as a three-character alphabetic abbreviation instead of a numeric value. This should eliminate the confusion over which comes first, the month or the day, in our old format of xx/xx/xx. The new format is as follows: dd mmm yy hh:mm:ss.
- The REX DUMP Service HEADER option will now call this REX service to obtain the system date and time for display.
- All system abort messages will now print with a date/time stamp.
- Options have been added to the MESSAGE, WAIT, ABORT, and KILL REX Services to print a date/time stamp with a message.

PORTINFO

The PORTINFO REX service is included in the system when a LINEMONITOR macro is specified in the SYSGEN. It was previously included only if WORKBENCH was specified. Then it was brought in with two other REX services.

PORTINFO is now used by several new OC Directives, TMP Directives, and JM overlays designed for use in a multi-batch environment.

DMP REQUEST HANDLING

The CLASSIC system has the capability of handling up to thirty-two DMP channels when executing in virtual mode. Prior to MAX IV Revision I.Ø the thirty-two DMP channels were handled with two tables. The two tables are now combined into one table.

The new table address is maintained in location #ØØ61. The reason for this change is the addition of the CLASSIC 32/85 system which used location #ØØ7Ø through #ØØ7F as mailboxes for the I/O Controller (IOC) subsystem(s).

Changes were limited to modules IS.DMP, IS.FTA, AD.INI, and ADU OC directive. It should be noted that the effect of this change is a decrease in processing time required for these modules.

OPERATOR COMMUNICATIONS REVISIONS

AVA — List available terminal ports.

BRO — Broadcast a message to all terminals.

CPD — Change Partition Definition

DATE — Set or display system date.
The DATE Directive now accepts input in either the old format of mm/dd/yy or in the new format of dd mmm yyy depending on whether the second parameter is alphabetic or numeric. It will always output date in the new format:

dd mmm yy.

ECM — Error Correcting Memory.
The ECM OC Directive has been modified to check LCBMST Bit 8 (C32) before processing the command. If C32 is set, the CPU is a CLASSIC 32/85. This means that error correcting memory is always enabled and that there is no memory interleaving. If the CPU is a CLASSIC 32/85, all ECM processing is bypassed and the following message is displayed: "CLASSIC 32/85 MEMORY ECM ALWAYS ENABLED"

GSR — The Global Shared Regions (GSR) OC Directive has been modified to indicate which virtual pages of shared memory shall bypass the cache when accessed.

FOL	File manager files offline.
MPI	Expanded task information.
MSG	Send a message to a terminal.
SRG	Display shared memory regions.
TCL	Terminal Control List.
TOL	Display TOCLOCK entries.
TRA	Transport responsibility.
WHO	Displays who is logged onto system.
WTH	What is happening with a task.

RESIDENT OC PACKAGE

A new package of Resident OC Directives is available. This package contains those OC directives that are supported by the MAGIC product. These are AVA, BRO, MSG and WHO.

DATA STRUCTURE REVISIONS

COLD START INITIALIZATION

Cold start now recognizes a CLASSIC 32/85 CPU type. It sets LCBMST Bit 8 (C32) to indicate this and sets up 8 large maps.

RESTART INITIALIZATION

Restart initialization checks LCBMST Bit 8 (C32) set by cold start. If the CPU is CLASSIC 32/85, it issues a WRITE memory status (WBS) instruction to indicate error correcting memory on and no memory interleave.

SHARED MEMORY CONFIGURATION

MAX IV Revision I.Ø supports a new privileged instruction that allows memory to be shared between CPUs in a multiple CPU (32/85) system. This new instruction (LBCI,Rx) identifies the starting address and the length of a shared region so that memory management units can properly handle their memory caches.

LBCI,Rx is executed in the TASKMASTER at the time the maps are loaded.

- It informs the CPU firmware that access to virtual pages VP through VP+L-1 in the operand map shall bypass the cache.
- The format of this instruction is identical to the format of the LOMP instruction.
- Bypassing of the cache is nullified when a LIMP or LOMP instruction referencing the same map is executed.

INSERT SHARED REGION

The insert shared region REX service checks to see if a global shared region is shared between CPUs (CLASSIC 32/85). If so, it sets up the virtual page and length of the region in the calling task's TCB word TCBBPC (bypassed cache). Also it will return an error on a task which tries to insert more than one bypass cache region at a time.

EXTRACT SHARED REGION

The extract shared region REX service was changed so that when a shared region that is set up in bypass cache is extracted, the task's TCB indicator TCB is also cleared.

TASK/OVERLAY CATALOGER REVISIONS

The Task/Overlay Cataloger (TOC) drops support JM and TMP of the CHANGE Directive. This function is now an OC overlay called CPD for Change Partition Definition.

The following two directives are now supported:

- SQUEEZE
- CLE (Command Line Extension)

SQU

The SQUEEZE Directive allows a loader module file to be compressed online by moving one module at a time in order to consolidate as much free space as possible.

CLE

The TOC processor now recognizes the CLE Directive and generates a TCB extension called Command Line Extension. This extension is 41 words long and the first word contains the length.

Directory manipulating directives lock out the load module file during manipulation of the directory when TOCLOCK is enabled. If TOCLOCK Ø is specified in Part 1 (the System Structure Block) of the SYSGEN, the lockout feature is NOT enabled.

TRANSCENDENTAL INSTRUCTIONS AND THE LEVEL 5 TRAP

The following changes have been made to the level 5 trap to support the transcendental instructions for MAX IV Revision I.Ø.

Upon entry to the trap the operation code of the trapped instruction is tested as follows:

- If it is a transcendental instruction, a branch is taken bypassing and floating point code to check for a user trap.

- If a user trap exists, a branch is taken to it and the return address is modified.

- If there is no user trap, the return address is incremented by one and a return through 14.EXR is taken.

Condition codes after the execution of an ordinary floating point instruction reflect the result of the instruction, not the conditions of the arguments.

Transcendental instructions have information provided through condition codes if the requested function could not be performed because of an out-of-range argument.

Example:

If an argument exception occurs, this is marked by setting of Condition Code Ø (CCØ) and the instruction being trapped by the level 5 trap.

CLASSIC 32/85 MEMORY TYPE

Type E memory is now defined as the memory of a CLASSIC 32/85.

TMP DIRECTIVES

The following is a list of TMP directives that have been either added or modified in MAX IV Revision I.Ø.

AVA	List available terminal ports
AVF	Advance file
AVR	Advance record
BAU	Display or change BAUD rate
BKF	Backspace file
BRO	Broadcast a message to all terminals
DEV	Display device information
FOL	File manager files offline
HOM	Home an I/O device
MSG	Send a message to a terminal
NUM	Numeric calculation
REW	Rewind a file
SYS	Display system identification and memory
TAS	Test file assignment
TER	Terminate I/O
WEO	Write end of file
WHO	Displays who is logged onto system
WTH	What is happening with a task

MAGIC TMP DIRECTIVES

TMP Directives AVA, BRO, MSG, and WHO are for use with the MAGIC product.

JM OVERLAYS

For the first time in MAX IV history JM overlays are being supplied as part of the system. The following is a list of JM overlays that have been added in MAX IV Revision I.Ø.

AVA	List available terminal ports
BRO	Broadcast a message to all terminals
BOX	Print a banner page before a listing
CPD	Change Partition Definition
FAT	Print a formatted list file assignments
FOL	File manager files offline
FSP	File manager volume space
MSG	Send a message to a terminal
TNA	Print a list of system disc transport
WHO	Displays who is logged onto system

MAGIC JM OVERLAYS

JM overlays AVA, BRO, BOX, MSG and WHO are for use with the MAGIC product.

TABLE OF CONTENTS

		Page
CHAPTER 1	OVERVIEW OF MAX IV	1
1.1	INTRODUCTION	1
1.1.1	MULTIPROGRAMMING	1
1.1.2	MULTIPROCESSING	2
1.1.3	PROGRAMMING CONSIDERATIONS	3
1.1.4	SYSTEM GENERATION (SYSGEN)	3
1.2	MAJOR SOFTWARE FEATURES AND COMPONENTS	5
1.3	INTERRUPT SUBROUTINES	5
1.3.1	DIRECTLY CONNECTED INTERRUPT PROGRAMS	5
1.3.2	STANDARD INTERRUPT PROGRAMS	6
1.4	EXECUTIVE SERVICES	9
1.4.1	DIRECT CALLS	9
1.4.2	FORTRAN CALLS	9
1.4.3	OPERATOR DIRECTIVES	10
1.4.4	JOB CONTROL DIRECTIVES	10
1.5	TASKS	10
1.5.1	ROOT TASKS	11
1.5.2	OVERLAYS	13
1.5.3	TASK SCHEDULING	14
1.6	PRIORITY LEVELS AND INFLUENCE LEVELS	14
1.7	DEDICATED RESOURCE ITEMS AND INFLUENCE LEVELS	15
1.8	REAL-TIME TASKS	16
1.9	BATCH-PROCESSING TASKS	16
1.9.1	JOB CONTROL	16
1.9.2	JOB PROCEDURES	17
1.9.3	SIMPLE OR EXPANDED JOB CONTROL EXAMPLES	17
1.9.4	JOB STREAMS	18
1.9.5	JOB STEPS	18
1.9.6	JOB INPUT SPOOLING	18
1.9.7	JOB OUTPUT SPOOLING	18
1.9.8	LOADING OF "UNCATALOGED" PROGRAMS	19
1.9.9	TRANSLATORS	19
1.9.10	UTILITIES	19
1.9.11	FILE MANAGER UTILITIES	20
1.9.12	USER-CODED BATCH PROCESSORS	20
1.10	REAL-TIME OUTPUT SPOOLING	20
1.11	BASIC INPUT/OUTPUT SYSTEM	21
1.11.1	I/O HANDLERS AND DEVICES	21
1.11.2	INTERRUPT-DRIVEN PERIPHERAL DEVICES	22
1.11.3	SYMBIONTS	22

1.11.4 BASIC I/O SYSTEM SERVICES 22
1.11.5 SYMBOLIC DATA TRANSFERS 24
1.11.6 BINARY DATA TRANSFERS 24

1.12 FILE MANAGEMENT SYSTEM 25

1.12.1 SECONDARY STORAGE FILES 25
1.12.2 FIXED PARTITIONS (EXTERNALLY-DESCRIBED FILES) 25
1.12.3 REAL FILES (INTERNALLY-DESCRIBED FILES) 26

1.13 THE SYSTEM LOADER AND LOAD MODULES 26

1.13.1 LOADER PROGRAM RECORDS (LPR'S) 27
1.13.2 MODULE OBJECT RECORDS (MOR'S) 27
1.13.3 RESIDENT LOAD MODULE DIRECTORIES 27

1.14 MEMORY RESOURCES AND PROTECTION 27

1.14.1 AUTOMATIC CHECKPOINTING (ROLL-IN/ROLL-OUT) .. 28
1.14.2 MANUAL CHECKPOINTING (REX ROL) 28
1.14.3 VIRTUAL ADDRESS SPACE PROTECTION 28
1.14.4 OPERATING SYSTEM PROTECTION 29

1.15 STANDARD SYSTEM REAL-TIME PROGRAMS 29

1.15.1 EXCEPTIONAL CONDITION (X) TASK 29
1.15.2 OPERATOR COMMUNICATION (OC) TASK 30
1.15.3 ADDING CUSTOM OPERATOR DIRECTIVES 31
1.15.4 INPUT SPOOLER 31
1.15.5 OUTPUT SPOOLER 31
1.15.6 USER-CODED SYSTEM TASKS 32

1.16 SYSTEM BATCH-PROCESSING PROGRAMS 32
1.17 USER-CODED BATCH-PROCESSORS 32

CHAPTER 2 HARDWARE REQUIREMENTS AND UTILIZATION 33

2.1 DEVICE REQUIREMENTS AND USE 33
2.2 MEMORY REQUIREMENTS AND USE 33

2.2.1 MAIN MEMORY 34
2.2.2 MEMORY EXPANSION LIMITS 34
2.2.3 SECONDARY STORAGE REQUIREMENTS 34
2.2.4 HARDWARE MEMORY MAPS 36
2.2.5 ACTUAL MEMORY 38
2.2.6 VIRTUAL MEMORY 40
2.2.7 NON-MAP 0 RESIDENT CODE 45

2.3 DISC LAYOUT 49

2.3.1 DISC CONTENTS 49
2.3.2 SYSTEM RESIDENCE PACK 49

2.4 MULTIPLE MAP UTILIZATION 52
2.5 MULTIPLE GENERAL REGISTER BLOCK UTILIZATION 52

2.6 HARDWARE CONTEXT SWITCHING 53
2.7 SOFTWARE (TASK) CONTEXT SWITCHING 55

CHAPTER 3 MEMORY MANAGEMENT FACILITIES 57

3.1 CLASSIC HARDWARE FACILITIES 57

3.1.1 MEMORY ORGANIZATION 57
3.1.2 MEMORY ADDRESSING 57
3.1.3 MEMORY ADDRESSING MODES 57
3.1.4 ACCESS RIGHTS 59
3.1.5 SHARED-PAGE INDICATOR 59
3.1.6 THE VIRTUAL MEMORY ACCESS PROCEDURE 60

3.2 MAX IV MEMORY MANAGEMENT DEFINITIONS 61
3.3 MEMORY MANAGEMENT OPERATIONS 65

3.3.1 LOCAL PAGE ACQUISITION/DISPOSAL 65
3.3.2 PAGE SHARING 65
3.3.3 ACCESS RIGHTS MODIFICATION 68

3.4 TASK ORGANIZATION FOR MEMORY MANAGEMENT 68

3.4.1 NUMBER OF MAPS 69
3.4.2 VIRTUAL ADDRESSING RANGE 69
3.4.3 PAGE SHARING 70
3.4.4 MAX III-COMPATIBLE ALLOCATION 70
3.4.5 TASK PUSH STACK SIZE/EXTENSIBILITY 70

3.5 MEMORY MANAGEMENT SERVICES 72

3.5.1 REX SERVICES 72
3.5.2 SERVICES PERFORMED BY THE MODULE LOADER 76

3.6 SYSTEM CONSIDERATIONS IN MEMORY MANAGEMENT 78

3.6.1 NUMBER OF MAPS USED BY THE TASK 78
3.6.2 MAP VIRTUAL ADDRESSING RANGES 78
3.6.3 LOCAL PAGE USAGE 79
3.6.4 SHARED REGION USAGE 80
3.6.5 TASK ROLLABILITY 80

3.7 THE ROLLER 81

3.7.1 MAP Ø RESIDENT ROLLER 82
3.7.2 NON-MAP Ø RESIDENT ROLLER 83
3.7.3 ROLLER MEMORY ALLOCATION ALGORITHM 83
3.7.4 ROLLER DYNAMIC RECONFIGURATION 85
3.7.5 ROLLER INFORMATIONAL MESSAGES 86
3.7.6 ROLLER ERROR MESSAGES 86

3.8 MEMORY INITIALIZATION 87
3.9 EXTENDED MEMORY ON A CLASSIC 78XY COMPUTER 87
3.10 CLASSIC 78XY COMPUTER SHARED MULTIPORT MEMORY 87

CHAPTER 4 INTERRUPT AND TRAP PROCESSOR 89

4.1 INTERRUPT LEVEL Ø - POWER FAILURE AND RECOVERY INTERRUPTS 89

4.1.1 POWER EVENT 90
4.1.2 CURRENT POWER STATE 91
4.1.3 USER ROUTINES 91
4.1.4 POWER EVENT PROCESSING 92
4.1.5 COLD START AND RESTART INITIALIZATION 93

4.2 INTERRUPT LEVEL 2 - SOFTWARE PROTECTION TRAP 93

4.2.1 STALL ALARM 93
4.2.2 SYSTEM HALTS 93
4.2.3 UNCONDITIONAL HALT 94
4.2.4 TRAP WITH ANOTHER INTERRUPT ACTIVE 94
4.2.5 SYSTEM PROTECTION HALT 94
4.2.6 NO TASK RUNNING HALT 94
4.2.7 TASK PROTECTION VIOLATION 94
4.2.8 STANDARD TASK PROTECTION TRAP RECOVERY 94
4.2.9 USER PROTECTION TRAP RECOVERY 95

4.3 LEVEL 4 - UNIMPLEMENTED INSTRUCTION TRAP 95

CHAPTER 5 TERMINAL MONITOR PROGRAM (TMP) 97

5.1 CUSTOMIZING TMP 97

5.1.1 SYSGEN 97
5.1.2 TASK VARIABLES 98
5.1.3 I/O OPERATIONS 100
5.1.4 CATALOG OPTIONS AND CONSIDERATIONS 100
5.1.5 U$COMM USER HOOK 101

5.2 INVOCATION AND SHUTDOWN OF TMP 102
5.3 ATTENTIVE TASK 102
5.4 ERROR MESSAGES 103
5.5 COMMANDS 103

ABORT, KILL 104
ASSIGN, DEFAULT 105
AVA 106
AVFILE 107
AVRECORD 108
BAU 109
BKFILE 110
BKRECORD 111
BRO 112
DEV 113
FOL 115
HOLD 116
HOME 117
MSG 118

NUM 119
OCP 120
OPTION 121
PAS 122
RESUME 123
REW 124
ROC 125
RUN 126
STOP 127
SYS 128
TAS 129
TER 130
WEOF 131
WHO 132
WTH 133

5.6 OVERLAYS 135

5.6.1 ENTRY TO OVERLAY 136
5.6.2 STACK 137
5.6.3 OVERLAY SERVICE TABLE 138
5.6.4 OVERLAY SERVICE ROUTINES 139

5.7 NAMING OVERLAYS 142
5.8 PROGRAMMING CONSIDERATIONS 143

CHAPTER 6 EVENT LOGGING

6.1 OVERVIEW 145
6.2 USAGE OF DATA FILES 146
6.3 SYSGEN STATEMENTS AND OC DIRECTIVES 147

6.3.1 EVENT LOGGING OPERATION 147
6.3.2 STORAGE MODE 147
6.3.3 EVENT LOGGING EXECUTION..... 148
6.3.4 OPERATOR NOTIFICATION 148
6.3.5 EVENT LOGGING FILES 149
6.3.6 LOGGING NODES 150
6.3.7 LOG POINTS 150
6.3.8 IMMEDIATE FORMATTING OF LOGGING EVENTS 150

6.4 EVENT LOGGING REPORT TASK 151

6.4.1 FILES 151
6.4.2 OPTIONS 152
6.4.3 REPORT DETAIL LINES 152
6.4.4 SPECIFIC EVENT INFORMATION 152
6.4.5 INFORMATION MESSAGES 154
6.4.6 ERROR MESSAGES 154

CHAPTER 7 INTERTASK COMMUNICATION SERVICE

7.1 OVERVIEW 157
7.2 USING ITC 158

7.2.1 OPEN VIRTUAL CIRCUIT 158
7.2.2 SENDING MESSAGES 159
7.2.3 RECEIVE MESSAGES 159
7.2.4 FLOW CONTROL 161

7.3 INCLUDING ITC IN A MAX IV OPERATING SYSTEM 161

7.3.1 SYSGEN REQUIREMENTS 161
7.3.2 TASK REQUIREMENTS 162
7.3.3 SERVER DEFINITIONS 162

7.4 ITC OPERATOR COMMUNICATIONS (OC) DIRECTIVES 168

7.4.1 MES - MESSAGE 168
7.4.2 PRT - PORT INFORMATION 169
7.4.3 SVR - SERVERS 169

7.5 INTERTASK COMMUNICATIONS SERVICES (#42) 170

VCOPEN 171
VCACCEPT 173
VCSEND 175
VCCLOSE 178
VCGET 180
MSGGET 182
VCENABLE 185
VCDISABLE 186
VCLINK 187
VCINFO 189
SDINFO 191
PRTDEF 193
SVRDEF 195
VCRESET 197

7.6 ITC USER STATE TRANSITIONS 198

CHAPTER 8 QUICK LOAD MODULES

8.1 DEFINITIONS 201
8.2 MODULE OBJECT RECORDS (MORs) 202
8.3 LOAD PROGRAM RECORDS (LPRs) 203
8.4 DIRECTORIED DATA FILE CONTAINING LOAD MODULES 203
8.5 LOAD PROGRAM INSTRUCTIONS 204

8.5.1 DEE -- DEFINE (MODULE) ENTRY POINT AND EXIT LOADING PROGRAM 205
8.5.2 UTI -- USE TASK I-SPACE MAP IMAGE FOR LOADING 207
8.5.3 UTO -- USE TASK O-SPACE MAP IMAGE FOR LOADING 208
8.5.4 DDL -- DEFINE DIRECTORY OF LOAD PROGRAM 208
8.5.5 RMO -- READ MODULE OBJECT RECORDS 210
8.5.6 RLP -- READ LOAD PROGRAM RECORD 211
8.5.7 SSO -- SET TASK'S SYSTEM OPTION WORD 211
8.5.8 SPO -- SET TASK'S PROGRAM OPTION WORD 212
8.5.9 SNV -- SET NAMED VARIABLES 212
8.5.10 DTV -- DELETE TASK VARIABLES 214
8.5.11 WAR -- WAIT AND RELOCATE MODULE CONTENTS 214

8.5.12 MAR -- MODIFY ACCESS RIGHTS 215
8.5.13 IGS -- INSERT GLOBALLY DEFINED SHARED PAGE REGION 217
8.5.14 EGS -- EXTRACT GLOBAL SHARED REGION(S) 218
8.5.15 CPR -- CREATE PRIVATE SHARED REGION 219
8.5.16 IPS -- INSERT PRIVATE SHARED REGION 220
8.5.17 EPS -- EXTRACT PRIVATE SHARED REGION(S) 221
8.5.18 ISM -- INSERT SHARED LOAD MODULE 222
8.5.19 ESM -- EXTRACT SHARED LOAD MODULE(S) 224
8.5.20 ERA -- ERASE ALL SHARED PAGE REGIONS 225
8.5.21 ALO -- ALLOCATE PRIVATE PAGES 226
8.5.22 DEA -- DEALLOCATE PRIVATE PAGES 228
8.5.23 DMR -- DEFINE MODULE'S VIRTUAL RANGE 228
8.5.24 FAR -- REASSIGN A FAT ENTRY 229
8.5.25 NOP -- NO OPERATION 230
8.5.26 SAC -- SET AREA CONSTANT 230
8.5.27 ICM -- INSERT COPYABLE LOAD MODULE 231
8.5.28 CTA -- CLEAR TEMPORARY FAT ASSIGNMENTS 232
8.5.29 SOB -- START/STOP OBJECT AND CHANGE MAP MODES 232
8.5.30 SPD -- SET PROLOGUE DEFINITIONS 233
8.5.31 DPS -- DEFINE PAGE SHARING EXTENSION 235
8.5.32 DCA -- DEFINE COMPATIBLE ALLOCATION EXTENSION 236
8.5.33 DVE -- DEFINE VARIABLES EXTENSION 236
8.5.34 DUC -- DEFINE TASK UTILIZATION LIMIT/COUNT .. 237
8.5.35 FAV -- DEFINE 1-N VACANT FAT ENTRIES 238
8.5.36 FAD -- DEFINE THE ASSIGNMENTS OF ONE FAT 238
8.5.37 DFL -- DEFINE THE LOGICAL FILE INDEX TABLE (LIST) EXTENSION 239
8.5.38 SPL -- DEFINE SYSTEM VIRTUAL PAGE LIST EXTENSION 240
8.5.39 UDC -- GO TO USER-DEFINED LOADER CODE 240
8.5.40 USR -- DEFINE A USER EXTENSION IN THE EXT OF A TASK 242
8.5.41 WKB -- DEFINE WORKBENCH EXTENSION WANTED 242
8.5.42 CLE -- DEFINE COMMAND LINE EXTENSION 243

8.6 MODULE LOADER ERROR MESSAGE CODES 244
8.7 MODULE LOADER TRACE 245
8.8 SEQUENTIAL LOADER OPTION 246
8.9 U$MLOA USER HOOK 246

APPENDIX A FEATURES IN THE MAX IV OPERATING SYSTEM 247
APPENDIX B STANDARD DEVICE AND CONTROLLER NAMES 249
APPENDIX C STANDARD GLOBAL LOGICAL FILE NAMES 261
APPENDIX D MAX IV STANDARD BATCH-PROCESSING LOGICAL FILES 265
APPENDIX E ERROR MESSAGES AND RECOVERY PROCEDURES 271
APPENDIX F FAULT ISOLATION PROCEDURE 283
APPENDIX G OVERLAY AND BUFFER CONTROL BLOCKS 285
APPENDIX H LOGGING NODE DATA 287
APPENDIX I ITC SYSTEM EQUATES 291
APPENDIX J REVISION H TO REVISION I CONVERSION NOTES 293

INDEX 297

LIST OF FIGURES

1-1 MAX IV Hardware/Software Priority Structure 8

2-1 MAX IV Basic Hardware Requirements 35
2-2 Conversion of Virtual Addresses to Actual Addresses 37
2-3 MAX IV Utilization of Actual Memory 39
2-4 MAP Ø Addressing Space 41
2-5 Addressing Space of a Typical User Task 43
2-6 Addressing Space of a Typical Split User Task 46
2-7 Typical Disc Layout 5Ø
2-8 Hardware Context Switching 54

LIST OF TABLES

3-1 Access Rights 59
3-2 Memory Management Definitions 61

4-1 Power Events 9Ø
4-2 Power Event Processing 92

7-1 ITC Transitions 198

CHAPTER 1
OVERVIEW OF MAX IV

This chapter explains the capabilities of the MAX IV Operating System. The major features are introduced individually.

1.1 INTRODUCTION

MAX IV is a task-oriented multiprogramming system that manages the execution of any practical combination of real-time and batch-processing programs. The operating system also provides a set of services for these programs, and includes I/O device support for essentially all standard peripheral devices that can be attached to the CLASSIC line of computers. The system supports a set of utilities and language processors for program preparation. MAX IV is a disc operating system, and is particularly useful for execution of large programs that address large resident or nonresident data files.

MAX IV can be configured in a variety of forms to support an unusually wide range of general purpose and dedicated user applications.

MAX IV includes all the proven features of the MAX III Operating System. MAX IV extends these capabilities and improves efficiency by calling upon new and unique features of the CLASSIC computers. These computers are designed specifically to improve the performance and lower the overhead of the Operating System software. In particular, MAX IV manages multiple hardware memory maps to:

- Solve the problems of memory fragmentation
- Provide natural hardware relocatability of programs
- Provide natural program protection
- Extend total system memory addressing capability

MAX IV also uses multiple blocks of general registers in a manner that is transparent to the programmer yet lowers system context switching overhead dramatically. Features such as multiple memory pools and memory partitions, that are found in the older operating systems, are not necessary in MAX IV. A single available memory pool serves all needs. Task priority controls usage of this memory. An optional roll-out scheme can be added to extend the main memory.

1.1.1 MULTIPROGRAMMING

Rarely can programs have continuous use of the CPU for more than short periods of time. Most programs become wait-bound quickly (usually while waiting for I/O operations to complete) -- leaving the CPU in an idle state. A computer can only work efficiently when several programs are available in main memory. (When one program becomes idle, there is a good chance that another can utilize the time and computing power that would otherwise be

wasted.) The term "multiprogramming" is used to describe this concurrent (but non-simultaneous) task execution on a single CPU.

The basic unit of multiprogramming is the "task". A task is a complete program unit that is sufficiently described to permit it to execute in a manner that is concurrent with and independent of other program elements. An interrupt subroutine is naturally a process -- but is usually called a "directly connected interrupt routine", since it is invoked automatically by the hardware. The term "task" is applied to those system and application programs that execute under the control of the MAX IV Operating System -- at software-generated interrupt levels called "task levels".

1.1.2 MULTIPROCESSING

When more than one hardware processor is used to simultaneously execute more than one program, the term "multiprocessing" is used. Actually, an element of multiprocessing can be realized in a single CLASSIC computer, since its high speed I/O channels (Direct Memory Processors (DMP)) have their own programs and access paths to memory. These paths can provide a useful level of parallel processing when the optional multi-port memory system is installed and when programs and I/O operations are being handled simultaneously in different memory modules.

The term "multiprocessing" is usually reserved for systems in which two or more complete computers operate in parallel -- for purposes of:

- Security (backup)
- Increasing raw computing power
- Sharing large data bases

A CLASSIC computer can be interfaced to other MODCOMP computers or computers of other vendors. MAX IV supports many such configurations -- with the computers tied together by communications lines, high-speed links, or shared-memory modules.

In shared-memory systems, enough private memory must be available in each computer to support the chosen operating system, and any shared-memory areas must be designated as data areas or global common areas -- through which all CPU-to-CPU communication is passed. These shared-memory modules must be placed at the highest actual memory addresses in the configuration. The best arrangement is to configure all private-memory modules at low actual addresses (near page Ø) and to configure all shared-memory modules at high actual addresses. This configuration permits easy expansion of both shared and private memory regions.

1.1.3 PROGRAMMING CONSIDERATIONS

REX service calls to perform I/O operations on disc devices under MAX IV returns an error condition.

Resident tasks in MAP Ø are restricted from addressing global common in MAX IV since such regions are not normally in MAP Ø. Such tasks, being privileged however, can address these common regions with special LOAD and STORE instructions without the regions being mapped. Resident tasks cannot dynamically allocate memory in MAP Ø other than in single page increments. This limitation avoids the need for a virtual space allocation scheme in MAP Ø, which would be subject to fragmentation problems similar to those found in systems where an addressing space must be shared by many independent users. To avoid such restrictions, the user can catalog tasks, that would otherwise execute in a resident state, as nonresident tasks. At system start-up such tasks can be "established" and made "non-rollable". In such cases they execute in their own addressing space yet can perform dynamic allocation, can address global common, and have all the advantages of a resident task. If it is necessary for such tasks to address the system nucleus, they can be cataloged to run in the privileged state. Any privileged task can address the nucleus addressing space (MAP Ø) by special instructions even though they can be resident in another addressing space.

The system generation statements of MAX IV are entirely different from those of earlier operating systems - although the system generation procedure is essentially the same. Most operator directives, Job Control statements, and system processor statements are basically the same although some minor differences exist. The MAX IV Task/Overlay Cataloger and the MAX IV Link Editor (M4EDIT) are notable exceptions to this rule.

1.1.4 SYSTEM GENERATION (SYSGEN)

System generation is described in the MAX IV SYSGEN, System Guide Manual listed in the Preface. Generation of the memory resident elements of an operating system are accomplished using the MAX IV MACRO Assembler (M5A) and the MAX IV Link Editor (EDI). The Stand-Alone Linking Loader can be substituted for the Link Editor (EDI) in cases where the linking time can be tolerated each time a fresh copy of the system must be loaded into main memory.

The nonresident (disc resident) elements of the operating system are prepared on disc with the MAX IV Task/Overlay Cataloger and other utilities.

The system generation process requires the user to write several specification statements in a higher-level MACRO language. These statements will permit the user to customize the MAX IV system -- including only those optional elements that are required for applications. Performing a system generation is done in four phases:

PHASE 1 -- Specify the desired resident elements in four parts with basic MACRO languages:

a) System Structure Elements
b) Basic I/O System Elements
c) Task Related Elements
d) File Manager Configuration

To these special assemblies, other user-specified elements can be supplied (resident REX Services, operator directives, resident task programs). The resulting group of object programs can be thought of collectively as the "main program" of the operating system and is called the SYSBLOCK. In most cases, these program elements consist of only simple data bases and linkages that "call" the appropriate elements from the MAX IV System Elements Library.

PHASE 2 -- Link the object elements produced in Phase 1 with the elements of the MAX IV System Elements Library, loading only the routines that were referenced.

PHASE 3 -- Load and initialize the system object image in main memory. A special Stand-Alone Loader (SAL) can be used to perform this function.

PHASE 4 -- Use the newly generated system to establish all nonresident elements.

1.2 MAJOR SOFTWARE FEATURES AND COMPONENTS

The following features and components of the MAX IV Operating System are discussed in this chapter:

- Interrupt Subroutines
- Executive Services
- Task types and scheduling
- Priority levels and Influence levels
- Dedicated Resource Items and Influence Levels
- Real Time Tasks
- Batch Processing Tasks
- Real Time Output Spooling
- Basic Input/Output System
- File Management System
- The System Loader and Load modules
- Memory Resources and Protection
- Standard System Real-Time Programs
- System Batch-Processing Programs
- User-Coded Batch Processors

For information on the data structures of MAX IV, refer to the MAX IV DATA STRUCTURES System Design Manual listed in the Preface.

Some of these components are passive, requiring that someone "call" them. Still others are active components, executing periodically or in response to certain hardware or software events. Some components are permanently resident in main memory, while others are loaded from secondary storage when needed.

1.3 INTERRUPT SUBROUTINES

Interrupt subroutine types are:

- Directly connected interrupt programs.
- Standard interrupt programs.

1.3.1 DIRECTLY CONNECTED INTERRUPT PROGRAMS

Some active MAX IV components are connected to hardware interrupt levels permanently. In most cases, these interrupt levels are enabled except during critical instruction sequences. All interrupt subroutines are permanently resident in main memory.

A user can add custom directly-connected components to unused interrupt levels if these programs must meet critical response time requirements, and can afford to operate above the operating system.

1.3.2 STANDARD INTERRUPT PROGRAMS

The following standard directly-connected interrupt programs are included in MAX IV:

LEVEL	NAME	PURPOSE
#Ø	Power Failure/Auto Restart Interrupt	Provides for orderly shutdown and restart of MAX IV during AC power cycling.
#1	Hardware Fault Trap	Detects, analyzes, and corrects memory memory parity errors.
#2	Program Exception Trap	Processes violations of memory access rights and privileged instructions.
#3	Multiprocessor Interrupt	Used for communications between other CPU's (optional); also used as a debug trace/monitor interrupt with the SHADOW.
#4	Unimplemented Instruction Trap	Processes entry of all Executive Service (REX) calls and optional instruction simulations. Provides an orderly transfer between unprivileged (program) and privileged (supervisor) execution states.
#5	Arithmetic Exception Trap	Processes floating-point overflow/ underflow violations and normalizes the results.
#6	Clock Interrupt	Updates critical timers directly and triggers a lower priority level (low clock) to process other less-critical system timers. Timers updated at this level include CPU Utilization Timer (current task only) and System CPU Utilization. The CPU Utilization bar graph and the memory viewer are also updated.
#7 . . . #B	External levels	Optionally used by MAX IV as task schedulers.
#C	I/O Data Interrupts (64 sublevels)	Processes data characters and words from slow I/O devices or processes End-of-Block signals from high-speed DMP devices. Each device controller channel has a unique interrupt subroutine entry, but similar channels can share a common handler.

#D	I/O Service Interrupts (64 sublevels)	Processes initiation and termination of each I/O operation and maintains queues of waiting operations to be performed by each controller channel. Each controller channel has a unique interrupt subroutine entry, but similar channels can share a common handler.
#E	Console Interrupt and Low Clock	Terminates and restarts the standard Operator Communication task. The level is also used to update the less critical timers deferred by level 6. Such timers include: o Task-scheduling timers o A delay timer fo each active task o I/O watchdog timers o The Time-of-Day Clock
#F	Taskmaster Interrupt	This level, being the lowest priority hardware level, multiplexes the non-interrupt time of the CPU into many multi-program levels (task levels). It also allocates some execution resources (a register block and one or two memory maps) to the currently executing task when control is given to it. The Taskmaster context-switches from one task to another when certain events occur that request the level. These events include: o Completion of a delayed or asynchronous service o Activation of a directly connected interrupt, such as a timer expiring or I/O service completion o Return of a system resource to a pool of resources o Direct request from another task

Figure 1-1 illustrates the relationship between the hardware interrupt priority levels and the software (task) priority levels created by the Taskmaster interrupt level. The task priority levels are below the interrupt levels. All references to hardware levels are hexadecimal notation (#Ø to #F) and all references to software task levels use decimal notation (Ø to 255).

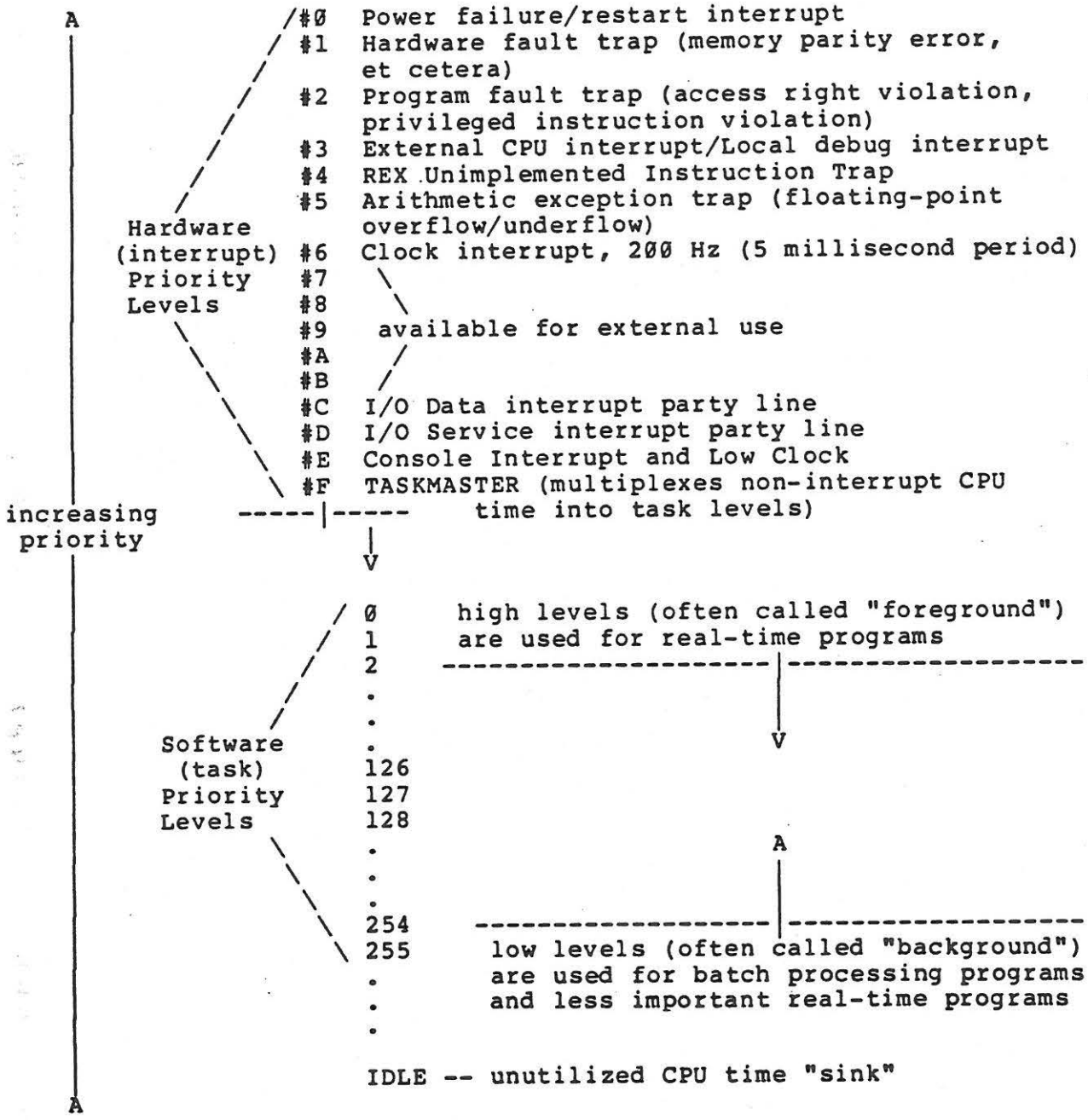

A
Hardware (interrupt) Priority Levels
#Ø Power failure/restart interrupt
#1 Hardware fault trap (memory parity error, et cetera)
#2 Program fault trap (access right violation, privileged instruction violation)
#3 External CPU interrupt/Local debug interrupt
#4 REX Unimplemented Instruction Trap
#5 Arithmetic exception trap (floating-point overflow/underflow)
#6 Clock interrupt, 200 Hz (5 millisecond period)
#7
#8
#9 available for external use
#A
#B
#C I/O Data interrupt party line
#D I/O Service interrupt party line
#E Console Interrupt and Low Clock
#F TASKMASTER (multiplexes non-interrupt CPU time into task levels)
increasing priority
Software (task) Priority Levels
Ø high levels (often called "foreground")
1 are used for real-time programs
2
126
127
128
254
255 low levels (often called "background")
are used for batch processing programs
and less important real-time programs
IDLE -- unutilized CPU time "sink"
A

Figure 1-1. MAX IV Hardware/Software Priority Structure

1.4 EXECUTIVE SERVICES

Reentrant Executive Service subroutines (REX services) are available to all tasks. These services permit the following functions to be requested.

- File Management Services
- I/O Operation Queuing Services
- Task Execution Control Services
- Task Scheduling Services
- Byte-String Syntax Analysis Services
- Code Conversion Services
- Information Services
- Utility Services

The usage and calling conventions of the MAX IV Executive Services are described in the MAX IV Executive (REX) Services, System Guide Manual, listed in the Preface. The subset of these services, which are also universally executable under MAX III is summarized in this publication and are fully described in the MAX III OPERATING SYSTEM Reference Manual listed in the Preface.

A user can code reentrant and recursive REX services and add them to the system at system generation time.

1.4.1 DIRECT CALLS

Executive services can be invoked in many ways. The most direct way is to execute the desired REX instruction in ASSEMBLY language:

EXAMPLE

```
LDI,R8     servicenumber
REX,MAXIV
```

Depending on the service, it may require that various calling arguments be pre-loaded into Registers or included in-line following the REX instruction. Arguments returned by the execution of the service may be in the Registers when the service returns to the calling program. REX services do not modify any Registers other than those used for passing or returning arguments. Refer to the MAX IV EXECUTIVE (REX) SERVICES System Guide Manual.

1.4.2 FORTRAN CALLS

The following format of calling Executive Services adopts FORTRAN subroutine conventions. (Calls exist for the ISA FORTRAN PROCESS I/O standard calls as well as other additional non-standard calls.)

```
CALL verb ([argument],...)
```

FORTRAN calls exist for all Executive Services, even though FORTRAN programs can use direct ASSEMBLY-language forms if desired by using INLINE/FINI statements. Refer to the MAX IV FORTRAN IV Language Reference Manual.

1.4.3 OPERATOR DIRECTIVES

The system operator can execute many Executive Services that exert control over other tasks by typing directives that are (in the following format) interpreted by the Operator Communication Task.

```
/taskname/verb  [argument]...
```

Refer to the MAX IV OPERATOR COMMUNICATIONS System Guide Manual.

1.4.4 JOB CONTROL DIRECTIVES

The batch-processing programmer can execute many Executive Services by entering Job Control Directives (at the beginning of a job or between job steps) in the following format:

```
$verb  [argument]...
```

These services can only affect resources that have been committed to batch-processing tasks. Refer to the MAX III/IV NON-RESIDENT JOB CONTROL AND BATCH FACILITIES Programmer's Reference Manual.

1.5 TASKS

Task programs can be assigned to 256 unique software priority levels. More than one task can execute at the same priority level, which forces them to share time with each other on a cyclic basis (if a system "ROUNDROBIN" feature is optioned). The maximum number of tasks that can exist in the operating system and be active at any one time is usually limited by main memory and secondary storage resources rather than the number of priority levels.

The highest priority task levels are designed specifically for the execution of important real-time tasks. These levels are sometimes called "foreground" levels. Level Ø is the highest level, though it is represented by the lowest number in absolute value.

The lower priority levels are designed specifically for the execution of less important real-time tasks and batch-processing tasks. These levels are sometimes called "background" levels. Level 255 is the lowest level, though it is represented by the highest number in absolute value.

In MAX IV there is no "magic" boundary that determines whether a particular task is "foreground" or "background." A continuous range of priorities is available for assignment of application programs. It is unlikely that 256 tasks will exist in a system, but the wide range of priorities is usually sufficient for new application programs to be installed.

In the MAX III Operating System, "middleground" levels are used to execute a range of lower priority tasks that are in a state of development. The tasks being debugged are given their own isolated memory pool so that a "hang" of such a task cannot compromise execution of critical real time programs -- due to fragmentation of memory resources. MAX IV does not need the concept of middle-ground, because its single memory pool can be managed (by using

hardware maps) in a way that prevents memory fragmentation. A system programmer need only install un-debugged tasks at priority levels below any critical tasks in order to ensure system integrity. Memory space will always be obtained by high priority tasks if enough space is configured in the system and belongs to lower priority rollable tasks.

Nonresident programs, which require contiguous memory, only need contiguous virtual addressing space in MAX IV. Memory maps permit MAX IV to construct contiguous virtual addressing spaces from noncontiguous actual memory pages.

1.5.1 ROOT TASKS

A Root task is a program that initially starts execution at a unique multiprogram level and is independent of other tasks. It requires a certain maximum amount of memory and I/O resources that are obtained for it when it is activated, and (optionally) are specified when it is cataloged.

Root tasks are cataloged by name in system task/overlay directories. These directories can be entirely on a disc or can be all or partially maintained in main memory.

Root tasks are classified further as to their size, residency, and execution mode as described below:

- o USER TASKS: operate in the unprivileged mode and are isolated from the operating system by being assigned a virtual addressing space (one or two hardware maps) into which only the actual memory areas that the task needs to access are included by the system. Such tasks are not allowed to execute privileged CLASSIC 78xy instructions but can communicate with the operating system through Executive (REX) services. Such tasks can communicate with other tasks by shared memory areas that the system maps into the virtual addressing space of each task that is to communicate. Permanent (global) or temporary (private) sharing areas in memory can be specified for this purpose.

  - Small User Tasks: use less than 16K words of memory at any one time and are assigned only one small virtual addressing space (small map) by the Taskmaster. If a small map is not available, an available large map is assigned.

  - Large User Tasks: use less than 64K words (and more than 16K words) and are assigned only one large virtual addressing space (large map).

  - Split User Tasks: use less than 128K words of memory at any one time, and their instructions must be separated from their operands so that neither exceeds 64K words. These tasks are assigned to two large virtual addressing spaces (large maps) -- one for operands and one for instructions.

User tasks that are lower in priority than critical real-time tasks can be debugged on-line. Such tasks should be used exclusively for the development of application programs unless the programs must execute privileged instructions.

- SYSTEM TASKS: operate in the Privileged mode. Such tasks can execute any CLASSIC 78xy instruction. Nonresident system tasks are mapped in unique virtual addressing spaces according to their size -- just as user tasks -- but resident system tasks are permanently mapped in the virtual addressing space of the operating system (MAP Ø). The experienced user can add system tasks to MAX IV, but such tasks cannot be debugged on-line unless critical real-time programs can tolerate possible system halts. Normally, system tasks are reserved for standard system programs that, according to priority requirements, can be allowed to compete with the user's application programs for CPU time. Examples include the Exceptional Condition Task, Operator Communication Task, and Spooler tasks.

- RESIDENT TASKS: have main memory resources assigned to them permanently at system generation (SYSGEN) time. System tasks can be resident if their response-time requirements are critical and if enough memory resources and MAP Ø addressing space are available to support them. All tasks that reside in MAP Ø are resident and privileged.

- TRANSIENT NON-RESIDENT TASKS: only use memory when active or scheduled. One memory page is used when they are scheduled, and the remaining memory resources are allocated when they are activated. They are then retrieved from special disc files and loaded into the allocated memory. Either user tasks or system tasks (which do not execute in MAP Ø) can be nonresident . When these tasks become inactive (they exit or are aborted), their memory resources are deallocated and their existence is not retained in the CPU queue unless they remain scheduled by a timer or an interrupt scheduler. Nonresident tasks are relocated by the hardware when loaded, because of the use of memory maps. These tasks are loaded quickly in a memory format requiring only two disc accesses in most cases.

- PRE-SCHEDULED TASKS: are nonresident tasks that retain their existence in the CPU queue -- even when they are not active and even when no timer or interrupt has scheduled them for activation. Such tasks are designated at system generation time.

- ESTABLISHED TASKS: are nonresident tasks that have been loaded and allocated resources, and then made to have the characteristics of resident tasks until otherwise requested. Either user tasks or system tasks can be established. When these tasks become inactive, their memory resources are retained, and only one copy of the task program is ever loaded. Such tasks are still rollable if their memory is needed by higher priority tasks but can be patched during the development stage since they are only loaded once. An established task can also be designated as unrollable but this practice should be minimized if true priority scheduling is to be maintained.

Established tasks and Pre-Scheduled Non-Resident tasks have the advantage of quicker access than Transient Non-Resident tasks -- yet retain the ease of replacement (recataloging) of nonresident tasks.

1.5.2 OVERLAYS

A root task has a maximum amount of virtual addressing space specified for it when it is cataloged. Some of this space is assigned actual memory when the root task is activated, while some of the actual memory can be assigned dynamically at runtime. If the program becomes too big to fit its virtual addressing space, or if the programmer wants to limit the amount of actual memory assigned to the task, an explicit overlay technique can be adopted. Overlays are programs that are loaded into the virtual addressing space of an already existing root task -- and which can overlay any or all of the instructions and data areas that are no longer needed by the root task.

Certain data areas (which are often called "common" areas or regions) can survive overlay loading so that they can permit communications between the root task and its various overlays or between one overlay and another. If an overlay completely overlays its root, communications between such programs must be accomplished by secondary storage files (for example, the various job steps of a batch-processing job are accomplished as a series of such overlays).

Overlays must share all the same resources and use the same priority level of their root task and can allocate only a limited number of resources that were established as a maximum for the root task.

If a task is established that uses overlays, it should be designated as a "batch" task so that its "root" program will be reloaded as an overlay by the system should the task exit or abort. The loading of an overlay can contract or expand to the amount of memory allocated to its root task -- but only to the extent that the root overlay program has been permitted use of a maximum addressing space when it was cataloged.

1.5.3 TASK SCHEDULING

Any task can be scheduled to be activated (started), resumed (continued after pause), or killed (abnormally terminated) by one or more components called Task Schedulers.

1.5.3.1 Task Scheduling Timers

Timers may be allocated by an Executive Service (CONNECT) from a pool of such timers. The expiration of these timers can be made to schedule a task. Such timers can be set in either of two formats:

- o Elapsed time (relative to when set)
- o Time-of-day (relative to midnight)

They may be scheduled as "one-shot" timers, which ensures that they automatically return to the pool once they have expired and performed their scheduling function, or as periodic timers, which allows them to expire and perform their scheduling function periodically and continuously until they are manually deallocated.

A single Delay Timer may be uniquely defined in the system for each task. These timers are similar to the Task Scheduling Timers in the timer pool, except they can only perform one function when they expire (resume execution) and cannot be made periodic. These timers provide a dedicated time base that the task can set and refer to as needed without affecting the task's execution.

1.5.3.2 Task Scheduling Interrupts

Unused external interrupts may be specified at system generation time for use in task scheduling. A reentrant Interrupt Service Routine is directly connected to these levels. Such levels may be allocated as task schedulers by the CONNECT service. The scheduled task becomes active, resumed, or killed at its software priority when this interrupt is requested externally (by hardware signal) or internally (by SIR command). These schedulers can also be made "one-shot" (requiring enabling each time) or "periodic" (requiring enabling only once).

One task can schedule another task directly with an Executive Service or by using interrupts or timers. The Operator Communication task has standard commands (directives) that permit it to schedule another task using a timer scheduler or an interrupt scheduler.

1.6 PRIORITY LEVELS AND INFLUENCE LEVELS

The execution priority level, assigned to a task by the system programmer, only determines the "winner" when a conflict exists between two or more tasks that are attempting to use a "basic" system resource such as:

- o Time
- o Memory
- o Register Blocks
- o Memory Maps

Unless further restrictions are imposed, any active task is still capable of affecting another active or inactive task, regardless of the individual execution priorities. If both tasks are user tasks, no real destruction can occur because the rules of privileged instructions and the natural isolation of hardware mapping prevents it. However, one task can activate, resume, kill, schedule, or modify the communication variables of another -- which indicates its capability to "affect" another task.

MAX IV has a separate inter-task control and cooperation scheme that can be invoked by a responsible system programmer as the task is cataloged or a system generation is performed. In the scheme, each task has a characteristic called "influence" that is similar to, but distinct from, priority. The absolute value of the influence number (a number between Ø and 255) determines a task's relative ability to perform a system function that affects another task. A task is allowed to affect another task only if they are "compatible". Compatibility exists when the affected task has a larger or equal influence level number (absolute value) than the affecting task. However, since the level of influence is inversely proportional to the level number, the compatibility criteria can be restated as: tasks are compatible when the affected task has an influence level that is lower than or equal to the influence level of the affecting task. To summarize:

Ø (highest level; lowest number)
.
.
.
255 (lowest level; highest number)

The only relationship between task priority and task influence occurs when the CHANGE priority service is executed. A task can be changed to a priority level no higher than the influence level of the task that requests the change. Also, priority and influence share the same inverse number range (Ø to 255).

1.7 DEDICATED RESOURCE ITEMS AND INFLUENCE LEVELS

Pools of resources that are sized at system generation time have individual items or groups of items that may be dedicated to tasks with compatible influence levels for example:

- Timers
- I/O Devices
- Global Shared Regions

Such resources can play the role of "the other task" and can be given influence levels just as tasks. When a task that has been cataloged with a low influence level tries to allocate resource items from such a pool and only items remain with higher influence levels, then the task is given a "pool empty" response even though dedicated resources remain in the pool for those tasks that request them with a sufficiently high influence level.

1.8 REAL-TIME TASKS

The only requirement for a real-time task is that it be installed at a sufficiently high priority level that it be capable of responding to the real-time events it must process. It is important that such tasks be installed by a system programmer who has an understanding of the interaction of several real-time tasks and the particular applications. In some operating systems these tasks are known as "foreground" tasks.

1.9 BATCH-PROCESSING TASKS

Batch-processing tasks are suitable for the execution of programs that can be started at an unpredictable time and can have their execution sustained for an unpredictable amount of time. Programs used for program preparation, system maintenance, and general data processing belong to this class of programs.

Batch-processing tasks actually operate in the "user" mode as if they were user-coded tasks. They may be privileged or unprivileged and isolated from the addressing space of the operating system (except by Executive Services).

The user is allowed to execute undebugged programs as batch processors (overlays of batch-processing tasks) in the unprivileged mode. Batch-processing programs are executed at low priority levels and are designated as rollable so that their memory resources and CPU time resources may be taken for use by higher priority tasks when necessary. In some operating systems these tasks are known as background tasks.

1.9.1 JOB CONTROL

The root task body (or initial overlay) of any standard batch-processing task is the nonresident Job Control program. This program acts as the original job initiator and establishes the initial memory and file resources for the first job step. It then overlays itself with the loading of the batch-processor program defined by the first job step. The Job Control program is also automatically loaded between all job steps so that it can establish appropriate memory and file resources at each step. The next job step is invoked by the current job step calling the EXIT service. Thus, the Job Control language facilitates the sequencing and resource control of all job steps through the termination of the job. At the end of a job, the batch-processing task seeks a new job or suspends itself if no suitable job can be found.

The Job Control language is a very flexible "macro-like" procedural language that permits the user to control those system resources set aside for batch-processing tasks at each job step.

1.9.2 JOB PROCEDURES

One of the most powerful features of Job Control is the ability to define a series of Job Control statements (and other batch-processor directive statements) as a procedure. Once defined, a procedure has the characteristics of a control language subroutine (or "macro") that can be called upon by name. These procedures can even have variable arguments (parameters) passed to them. Procedures can call other procedures to any level of nesting and have conditional looping and testing facilities. With this feature, a library of frequently used Job Control procedures can be cataloged on disc for instant access by call. This is of considerable value when people with a wide variation in experience levels must share the resources of the computer system. Inexperienced users can be taught to use only a limited set of convenient (but restrictive) procedures, which limit their access to the machine.

The following examples illustrate the difference between a simple procedure call and its basic Job Control language equivalent. The complete Job Control language is described fully in the publication entitled: MAX III/IV NONRESIDENT JOB CONTROL AND BATCH FACILITIES, Programmer's Reference Manual listed in the Preface.

1.9.3 SIMPLE OR EXPANDED JOB CONTROL EXAMPLES

A typical Job Control language procedure:

```
$JOB - JOHN BRAVE
$ASSIGN SI=CR SO=SCB
$EXECUTE FR5
C - TYPICAL FORTRAN PROGRAM
.
.
.
$$
$WEOF SO
$ASSIGN SI=SCB BO=SCA
$REWIND SI,BO
$EXECUTE M5A,,NOLO
$WEOF BO
$ASSIGN BI=SCA BO=SCB
$REWIND BI,BO
$EXECUTE M4E,,NOMAP
EDIT MAIN,BI
EXIT
$WEOF BO
$REWIND BO
$EXE TOC
FILE LM
OVE JOHN
CAT
EXI
$EXECUTE JOHN LM
```

1.9.4 JOB STREAMS

Batch-processing tasks take all their directions from job streams. Job streams are sequential files of data consisting mostly of Job Control Directives or procedures. The user's data to be processed can also be embedded in job streams or located on other data files. The job stream may be considered to be endless and consisting of many independent and unrelated jobs. Each job consists of one or more job steps. Job steps are related to each other and must be performed sequentially as entered. Individual jobs are assumed to be unrelated and may be executed (at the system's choice) in any order -- depending upon size, priority, and availability of resources.

1.9.5 JOB STEPS

A job step starts with the execution of the Job Control program that establishes the initial amount of memory required for the step and opens all files to be used by the step. This is followed by the overlay loading and execution of one or more batch-processing programs. When any batch processor calls the EXIT service, the step is terminated. The Job Control program is reloaded as a chain overlay to pass control to the next step. If the batch processor is forced to abnormally terminate (ABORT), then remaining job steps are bypassed and the entire job terminated. The batch-processing task then seeks another job.

1.9.6 JOB INPUT SPOOLING

When the optional input spooling task is included, all jobs may be read from a job entry device and organized into separate disc files by job. The input spooling function isolates the operator's input of jobs from the processing of these jobs so that execution can proceed at higher speed and with higher throughput.

1.9.7 JOB OUTPUT SPOOLING

Output spooling is a technique that allows one or more devices to serve the needs of many concurrent batch-processing real-time tasks. (The Job Output Spooling Task only supports printing devices.) The technique makes a program use a high-speed disc file as a buffer so jobs can complete their processing quickly and exit even though the physical output device has not yet received the information.

Spooling serves to increase throughput by keeping devices busy and by isolating processing from manual interventions (such as paper changing and device loading).

The output spooler is actually a symbiont task operating at a system task level. This task routes output device requests to disc by its own memory buffers and eventually outputs the disc resident information to the desired target devices. Printing information is compressed/uncompressed as it is transferred to/from disc in order to reduce the number of disc accesses and to save disc space.

1.9.8 LOADING OF UNCATALOGED PROGRAMS

The MAX IV Load Module Retriever can only load modules cataloged in a quick-load format by the MAX IV Task/Overlay Cataloger. When the loading of uncataloged programs or incomplete object programs is desired, and the time required to link-edit and catalog such programs is considered undesirable, special overlays of Job Control are loaded into the addressing space of a batch-processing task. These special loaders process a sequential object module and load it or link it in the same addressing space. Enough addressing space must be available to contain:

- The loader overlay
- The loaders symbol table (if linking is necessary)
- The program being loaded

The object format produced by the MAX III/IV Macro Assembler or the format produced by the MAX III Disc Cataloger can be processed by these loaders. These loaders are not capable of resolving the usage of global shared regions of memory by the programs that they load. Full link-editing and cataloging is necessary to accomplish this usage. Such programs can explicitly call the Executive Services that insert and delete global and private shared regions, as soon as they are loaded.

1.9.9 TRANSLATORS

The following computer language translators are available as standard batch-processing programs under MAX IV. These programs are requested by Job Control directives:

- MAX III/IV FORTRAN IV Compiler (16-bit integers)
- MAX IV FORTRAN IV Compiler (32-bit integers)
- MAX III/IV Macro Assembler

1.9.10 UTILITIES

The following multi-function programs are available as batch-processors under MAX IV:

- MAX III Link Editor -- a link editor capable of structuring complete programs in single virtual addressing spaces.

- MAX IV Link Editor -- a full-service link editor capable of structuring programs in multiple virtual addressing spaces, assigning attributes and access rights, and segmenting large programs into "tree-like" overlays.

- The Task/Overlay Cataloger -- places fully-edited programs on disc-resident data files for subsequent action by the system's loader. These programs may be designated as tasks or as overlays of existing tasks. Programs may be cataloged in quick-load absolute format or may be made relocatable.

- MAX III/IV Library Update -- an object library maintenance utility.

- MAX III/IV Source Editor -- a utility for source program preparation and maintenance for sequential or directoried files.

- MAX III/IV Link Loader-- a system processor that loads a main object module into memory, together with any additional external object modules, and links the modules into an executable set of machine code.

- MAX III/IV Direct Access Maintenance Processor -- a system processor that provides the utilities to manipulate FORTRAN direct-access directories.

- MAX III/IV Moving-Head Disc Preparation Program -- a system processor used to initialize an unused disc pack.

- MAX III/IV Binary Record Dump -- a system processor used to display records written in MODCOMP Standard Binary Mode.

- MAX IV Dump Analyzer -- a system processor that reads, formats, and analyzes the memory image and outputs information to either a terminal or a line printer.

- MAX IV Alternate Track Processor -- a system processor that supports the hardware implementation of alternate tracking. It provides an online capability of updating the MODCOMP Error Map.

1.9.11 FILE MANAGER UTILITIES

FMPREP - File Manager Volume Preparation Utility
FMLIST - File Manager Listing Utility

1.9.12 USER-CODED BATCH PROCESSORS

The user may code and execute non-real-time programs as overlays of batch-processing tasks (batch processors).

1.10 REAL-TIME OUTPUT SPOOLING

The output spooler not only increases the throughput of batch-processing tasks but may also be used by real-time programs to minimize the time required for them to stay in memory to write short reports. The actual reports produced by such tasks are rarely required to be available in real time -- as long as the task that produced the report is executed in real time.

1.11 BASIC INPUT/OUTPUT SYSTEM

The MAX IV Basic I/O System (BIOS) consists of two parts:

- Interrupt-driven or DMP-driven handlers exist for all physical I/O devices. These handlers are directly-connected to the I/O channels they control. The user need not interface to them directly.

- A centralized Basic I/O System, addressable as a series of Executive Services (REX calls), allows a user to interface to device handlers indirectly -- by way of logical files. Logical files are assigned to devices at task cataloging time, or at run time. These services permit a programmer to talk to the file's assigned device only in terms of physical records (for example, card, block, or sector).

1.11.1 I/O HANDLERS AND DEVICES

Devices may be real, logical (subdivisions of a disc medium such as real files), or imaginary (simulated by a task called a symbiont). Some devices have their own independent controller channel, while others must share a common controller channel. All devices are identified by name in the Logical Device Table.

The interrupt-driven handlers permit I/O operations to be performed concurrently (overlapped) with the execution of the programs that request the operations. Slow devices use data interrupts to transmit data bytes/words; higher speed devices use Direct Memory Processors (DMP's) to reduce interrupt overhead.

Service interrupts are used by all devices for initiating and terminating all operations. These interrupts manage the queue of operations waiting to be started on a particular controller channel.

Device error recovery may be automatic (retry initiated by service interrupt with unrecoverable errors reported to the system operator) or may be under the complete control of the calling task. The latter method allows a critical task to perform suitable application-dependent error recovery techniques, including device substitution.

Device handlers have both device-independent and device-dependent modes of operation. The former mode lets the handler simulate (if necessary) a series of device characteristics that are common to a class of similar (but non-identical) devices. The latter mode allows a programmer to use the unique characteristics of a particular device if desired.

1.11.2 INTERRUPT-DRIVEN PERIPHERAL DEVICES

Many standard I/O devices are supported by the MAX IV Basic I/O System. These include:

- Interactive Terminals (CRTs and hard copy devices)
- Card Readers
- Card Punches
- Paper Tape Readers
- Paper Tape Punches
- High-Speed Line Printers
- Slow- and Medium-Speed (one character at a time) printers
- Raster-Type Plotter/Printers
- X-Y Plotter
- Magnetic Tapes (9 track, NRZ and phase-encoded)
- Floppy Discs
- Cartridge Moving-Head Discs
- Multi-Platter Moving-Head Discs
- Large-Capacity Moving-Head Discs 4138
- Fixed-Head Discs
- Memory Plus (simulates a Fixed-Head disc except uses memory for storage)
- Communication Channels (Asynchronous and Synchronous)

1.11.3 SYMBIONTS

Unusual devices, which have difficult or unusual control protocol or perform functions unavailable in real devices, may be interfaced to the system by a symbiont. These symbionts are simply tasks that present an imaginary device interface to the programmer while insulating the user from the peculiar protocol of the real device or communications channel. Typical symbionts include:

- Job Input Spooler
- Job Output Spooler
- Remote CPU Interfaces (such as 2780, 3780, or 200 User Terminal)
- X-Y Plotter Spooler

1.11.4 BASIC I/O SYSTEM SERVICES

The user calls the I/O system by a series of Executive (REX) Services that all converge on a large central scheduling routine. This routine executes at the calling task level until an I/O operation is scheduled in the queue of the appropriate I/O device controller. These services schedule such functions as READ, WRITE, REWIND, and WEOF.

Each task has a local logical file list called the task's File Assignment Table (FAT). It is located in the task's TCB. MAX IV has a similar list of logical files, called the Global File Assignment Table (GFAT), that is kept in the TCB of the standard Exceptional Condition (X) task. These tables are used by the Basic I/O System to find the assigned device when an I/O operation is requested for a particular logical file by a program. The logical

files exist only as linked entries in these lists. The task's File Assign Table (FAT) is always used unless the desired file cannot be found in it. In this case, the Global File Assign Table (GFAT) is used. With this scheme, duplicate logical files may exist for different tasks -- and yet unique device assignments can be maintained. Likewise, a single logical file name may be used for communications with a device by many different tasks.

Logical files may be assigned to other logical files, but the final assignment must be to a device (real, logical, or imaginary) that must be in the System Logical Device Table. Disc devices (not files) may be kept permanently in this table or may be added and deleted with File Manager services, Job Control Directives, or special batch-processing utility programs.

Each logical file entry has two assignments called the current assignment and the default assignment. With this feature, a task may change its file/device assignments at run time and restore original assignments when desired. This is particularly useful for batch-processing tasks -- where each new job can be given preset standard file assignments by using all the default assignments.

Each logical file has a name. The programmer can address the file only by its name.

All I/O operations that cannot be started immediately are queued so that no program delay results due to a busy controller channel. Operation nodes in device controller queues are ordered by calling task priority and by chronological order for like task priorities.

Devices that share a common controller channel have "slough" queues that prevent unmounted or unavailable devices from stalling the central controller channel queue.

When using the Basic I/O System directly, the user must expect to deal with the assigned device in terms of discrete physical records or blocks. Buffers must be supplied in the user's program that are capable of containing exactly one physical record. Each device has a different physical record size; therefore, the user must often determine device size and adjust the program accordingly. The largest common buffer size for most standard devices does not exceed 128 words (1/2 memory page), although some special devices may be programmed with much larger buffers if the program is capable of using memory resources to such an extent.

Data is transmitted to and from memory data buffers within the program body of the calling task. In a FORTRAN program, which uses subroutines to define access methods, these data buffers may be transparent to the user; but are allocated and used by the FORTRAN Run-Time Package and reside in a labeled common area generated by the FORTRAN IV compiler or in an area near the end of the task body.

1.11.5 SYMBOLIC DATA TRANSFERS

Symbolic data is always transmitted to and from memory buffers in ASCII code by the Basic I/O System. This is the adopted international code used by MAX operating systems for symbolic transmissions. The code set, which consists of 128 defined data and control characters, requires 7 bits to define each character. Each character resides in an 8-bit memory byte with the most significant bit unused. Two such bytes reside in each 16-bit memory location. A complete description of symbolic data buffer formats is included in the MAX IV INPUT/OUTPUT, System Guide Manual, listed in the Preface.

If a particular device uses an external hardware code set other than ASCII, the device handler usually converts the internal codes to the desired code by either hardware means (ROM table conversion) or software means (indexed table look-up). The description of each handler in the publication mentioned above, describes whether translation is performed for a particular device. In some cases, code conversion must be done by the programmer before/after WRITE/READ operations are performed. In other cases, it may be easy to substitute a modified conversion table at system generation. It is also possible that a symbiont task can be written that performs code conversion when the original task addresses a particular symbiont device.

Symbolic data records that go to printing devices must contain control data that "tell" the device handler how to directly position the carriage for each WRITE operation. This carriage control data is usually transmitted in byte Ø of the programmer's data buffer. Devices that do not print usually ignore such data but transmit it as normal data to the device. Printing devices interpret this data and convert it into carriage positioning commands or special byte prefix sequences before the print data is transmitted. The carriage control data itself is not transmitted to a printing device. Often byte 1 is also involved as carriage control data when it is desired to defer the start of printed text to byte 2 (next word boundary).

For non-printing devices, a programmer's data buffer must always be arranged so that the left-most byte (byte Ø) in word Ø is the first transmitted byte. For DMP devices, an even number of bytes may have to be be transmitted because of the 16-bit word nature of some devices. A trailing pad byte (blank for symbolic modes, null for binary modes) is often transmitted in this case.

1.11.6 BINARY DATA TRANSFERS

Non-symbolic data is transmitted bit-for-bit from primary memory to device in most cases. If the target device is byte oriented, one 8-bit data byte in memory is transmitted to/from the 8-bit wide data path of the device. Two such bytes reside in each 16-bit memory location of the programmer's data buffer. The Standard Binary record format, as it appears in a programmer's buffer, is described in the MAX IV INPUT/OUTPUT, System Guide Manual listed in the Preface. This record format requires that bytes Ø, 2, and 3 be

reserved for control purposes so that the records can be identified as binary records and so that variable-length binary records can be defined by control bytes within the unit record itself. Most handlers have a Non-Standard Binary mode if these buffer prefixes are not desired. However, most MODCOMP standard software uses the Standard Binary format for non-symbolic data transmission.

Device handlers interpret binary data in various ways. Secondary storage devices usually transmit it unmodified. Some devices encode/decode it into bit patterns appropriate to the device. Other devices, such as plotting devices, interpret such data as functional commands.

1.12 FILE MANAGEMENT SYSTEM

The File Manager only concerns itself with the definition and use of collections of physical records and the utilization of space on secondary storage volumes.

1.12.1 SECONDARY STORAGE FILES

A disc pack (volume) is a basic subdivision of a disc controller. If the volume is removable, then the available secondary storage data base of the computer system extends beyond the number of physical disc transports available.

Any contiguous or noncontiguous subdivisions of a volume may be established as a named logical disc device in MAX IV. Such "devices" can be obtained in several ways. The File Manager calls these devices "real files." Any program can use these "devices" by assigning them to its logical files.

1.12.2 FIXED PARTITIONS - EXTERNALLY-DESCRIBED FILES

At system generation time, one or more fixed partitions of a disc Transport may be defined. These partitions are given unique device names and may be included as permanent entries in the System Device Table. Such devices can be associated with any logical file and used directly by the programmer through the services of the Basic I/O System. These devices define contiguous single extents of disc medium that are fixed to the transport regardless of which pack is mounted. For this reason, the File Manager calls them "externally described files". These devices may be addressed sequentially in a manner similar to a magnetic tape transport or directly by a relative position index (16 or 32 bits). Each partition may be addressed independently of others. Such devices are sufficient for use with fixed-head discs and for parts of the System Residence Volume but are not sufficiently flexible when many volumes can be alternately mounted on the same transport. These devices are provided so that the MAX IV Operating System and the MAX III Operating System can interchange secondary storage data files.

1.12.3 REAL FILES - INTERNALLY-DESCRIBED FILES

Real files are defined by directories and other data structures written directly on the volume medium. Whether these volumes are mounted or removed, the definition of their files remains as part of the written medium. (Secondary storage volumes that contain such files are said to contain "internally described files".) When a volume is mounted, all the files described on it may be potentially opened by name as they are needed by programs. When such files are no longer needed, they are closed. When a volume is dismounted, all of its currently open files must be closed before the dismount is permitted. The opening of files that exist on unmounted volumes causes operator messages to be generated that request mounting of volumes.

Real files are defined on volumes such as removable disc packs. Disc files may be defined or expanded on contiguous or non-contiguous sections of disc media.

Utilities (and optional Executive Services) permit the space on mounted disc volumes to be managed by batch-processing programs (or by real-time tasks). New files may be added or expanded as needed on mounted volumes. Old files may be reduced in size or destroyed. Files that have become fragmented on the media may be compressed (made contiguous).

Once a real file is opened for use by a program, it can be operated on with the services of the Basic I/O System. These services let the program deal with the file as a series of physical records (blocks or sectors). Both "block sequential" and "block direct" access methods are available for disc files within the basic I/O system.

The File Manager also permits the simple definitions of disc partitions to exist on a volume under its control, which allows definitions to change with volume mounting.

1.13 THE SYSTEM LOADER AND LOAD MODULES

Nonresident overlays and tasks are cataloged on disc in a special load module format that permits these programs to be brought into memory in large, single blocks -- often requiring only two disc accesses. The load module is in a format that requires no software relocation if the program has been cataloged in a normal manner with the MAX IV Task/Overlay Cataloger. The loader utilizes the natural hardware relocation benefits of memory mapping. The absolute locations of all nonresident programs, global commons, and reentrant shared libraries are determined at link-edit time or cataloging time. For special applications, a module may still be cataloged so that it can be relocated at load time, but this is rarely necessary in a MAX IV system.

Each load module of a root task or overlay consists of two types of records called Load Program Records (LPR's) and Module Object Records (MOR's). The "average" program has one LPR (a disc sector) and one MOR (a multi-sector record). Pointers to each of these items are included in the Load Module Directory (LMD) of the cataloged program load module file.

1.13.1 LOAD PROGRAM RECORDS -- LPR'S

A Load Program Record contains information (called "resources") for the program to be loaded. The system uses this information to select the appropriate memory and space environment of the program to be loaded (if not already allocated). For a root task, the LPR also contains the task's resource assignments that tell the Load Module Retriever how to build a TCB for the task when it is activated and becomes an independent multiprogramming entity.

The LPR also contains a "string chain" that specifies to the loader how it should read the body of the program. This chain facilitates the "scatter" read of the program even when not contiguous in memory. The chain is in a format that is directly interpretable by the Input/Output Processor of the CLASSIC 78xy computer.

1.13.2 MODULE OBJECT RECORDS -- MOR'S

The body of a nonresident program consists of one or more program extents called Module Object Records that are defined in the LPR's. These MOR's are read into the program's allocated memory area using a scatter-read technique as specified by the string chain in the LPR's. Only one disc access is normally required to complete the transfer.

1.13.3 RESIDENT LOAD MODULE DIRECTORIES

All load module files have their directory of cataloged programs as the first data item on the file. For critical program storage files, a resident directory may be defined in the system nucleus, which keeps resident copies of all or part of the file's load module directory for immediate access by the Load Module Retriever.

1.14 MEMORY RESOURCES AND PROTECTION

The allocation of actual memory pages to pages of virtual addressing space is managed by certain Executive Services. Any number of actual memory pages (256 words per page) scattered throughout actual memory may be put together to form any required contiguous virtual space for use by a program.

Because this feature eliminates a common memory fragmentation problem, such techniques as multiple memory pools and fixed memory partitions are unnecessary compromises for MAX IV. A single Available Memory Pool is all that is necessary for MAX IV to service the many asynchronous demands for memory space.

1.14.1 AUTOMATIC CHECKPOINTING - ROLL-IN/ROLL-OUT

The ability of a task to retain its main memory resources is determined by the task's execution priority level. Since batch-processing tasks and real-time tasks under development execute at low priority levels, their main memory resources are subject to be taken away at any time -- if higher priority tasks need these resources. Real-time tasks under development should be executed at priority levels higher than batch-processing levels, as a general rule. This ensures that the non-critical real-time tasks are the last tasks to lose their memory when memory becomes crowded, and the batch-processing tasks are the "first-to-go".

When any task needs memory and no memory can be found in the Available Memory Pool (AMP), then the "memory allocator" queues the request to the Roller task. This task attempts to find the lowest priority task that is designated as being rollable and has enough memory pages to satisfy the demand. The Roller executes as a resident system task (ROL) but has the ability to move its effective execution priority level in the CPU queue to a level appropriate to service the highest priority task needing memory. When the available memory pool is capable of serving all task memory requirements, the ROL task moves itself to the bottom of the CPU queue and stays there until some task needs more memory than is available in the available memory pool.

1.14.2 MANUAL CHECKPOINTING - REX ROLL

The ROLL REX service can override the above mentioned "Automatic Checkpointing" algorithm. Using this service a named task can be requested to be rolled out of main memory or to be rolled back into main memory. It can also change the rollability of a task. When a task is requested to be rolled out of main memory it stays rolled out until a REX ROLL (in) service request occurs.

1.14.3 VIRTUAL ADDRESS SPACE PROTECTION

Memory protection is provided by the memory maps. It should actually be called "virtual address space protection", since it permits each virtual page to be given a unique memory "access right" as to its usage by a particular program. The access rights of a virtual page may be one of the following:

Code	Restriction
'00	No access (nonexistent memory)
'01	Read only (protected data)
'10	Read and Execute only (protected program)
'11	Read, Execute, and Write (no restrictions)

This protection scheme causes a trap if any program violates the access rights of any addressing regions assigned to it. The trap may be used to abnormally terminate the program -- or made to return such attempts to a special routine in the program for processing as it chooses.

The advantage of this virtual access right protection over actual memory page protection is that pages of memory shared by several tasks can be given different access rights, which permits some programs to produce data that other tasks can access -- but not modify. The access rights may be changed dynamically to permit tasks to communicate through common areas without interaction problems.

Programs that separate their modifiable data areas from non-modifiable instructions are rewarded with valuable debugging aids that catch program failures before the evidence of what caused the failure is lost.

All user tasks are protected from each other as well as being isolated from the system. (Refer to the MEMORY MANAGEMENT FACILITIES chapter of this manual.)

1.14.4 OPERATING SYSTEM PROTECTION

Major elements of the operating system are protected from other elements of the operating system, which makes debugging new privileged system elements (I/O handlers, REX services, system tasks) easier and less prone to software failure.

1.15 STANDARD SYSTEM REAL-TIME PROGRAMS

Several MAX IV program elements are active task programs that compete for CPU time and other system resources with the user's own task programs. These tasks are placed in the priority structure in a manner that is advantageous to system performance.

1.15.1 EXCEPTIONAL CONDITION (X) TASK

This resident system task operates at the highest task priority level (Ø). Its purpose is to report exceptionl system conditions to the system operator and to perform task-level operations for the MAX IV system itself. When necessary the X task:

- Types various exceptional condition reports requested by direct interrupt driven elements of the system that cannot queue up messages themselves.
- Reports device and memory failures and off-line conditions to the system operator.
- Reports tasks that exceed their cataloged time limits.
- Performs system start-up and initialization functions including the initialization of resident directories for disc files.
- Performs user-coded system restart procedures.

- Interrupts operator directive entries if important messages must be typed on the operator's terminal printer.
- Updates calendar tables.
- Performs the first phase of the volume recognition sequence.
- Updates the Task Queue if the ROUNDROBIN scheduler is optioned.
- Performs type zero disc data structure updates.

1.15.2 OPERATOR COMMUNICATION (OC) TASK

This resident system task operates at a high task priority level to ensure that it has control over all tasks that need its services. Generally, this task's execution is suspended at all times except when the operator has requested to communicate with the system or when the system's Exceptional Condition Task (X Task) has informed the operator of exceptional conditions that require response.

The task is activated manually (actually it is aborted and restarted) by pressing the CONSOLE INTERRUPT switch or pressing BREAK or CONTROL "A" on the operator's terminal. It identifies itself, queues up a READ to the operator's terminal, and then usually relinquishes time to all lower priority tasks until the first character of an operator directive (command sentence) is typed by the operator. If approximately 10 seconds expire without such a response, the READ terminates and the task suspends itself.

If the system must type emergency information to inform the operator, the Exceptional Condition Task queues the message and internally requests the operator task if its response is necessary. If the operator task is already active because it is waiting for an operator entry or is performing a directive function unrelated to the emergency, it is aborted and restarted after a suitable time delay so that the emergency condition(s) can be reported. After the first character of a directive is received, the task takes exclusive-use of the terminal device until the next READ Directive is queued or when the task suspends itself.

When a directive is received from the operator, the task examines the first keyword of the command sentence to see if it matches an entry in a nonresident directory. If the desired operator communications directive processor is found in the nonresident directory, it is loaded as an overlay into a fixed buffer and executed. Long directive processors may consist of two or more loaded and executed overlay routines. Control then returns to the operator so that a new directive may be entered. If a new directive is not received within the time limit, or if the directive is a terminating directive, the operator task will be suspended. (The execution of some directives causes the operator task to become inactive (suspended immediately after they are executed) requiring that the operator again press the CONSOLE

INTERRUPT switch, or BREAK or CONTROL "A" when the services of this task are needed. Such directives are called "terminating" directives.)

The Operator Communication Task is described fully in the MAX IV OPERATOR COMMUNICATIONS, System Guide Manual, listed in the Preface.

1.15.3 ADDING CUSTOM OPERATOR DIRECTIVES

A user can add custom operator directives to the operating system as long as the first keyword (up to three characters) of the directive is unique. Such directives may be programmed so that they address the currently attentive task, the system as a whole, or a particular task only.

These directives may be added to the operating system at run time by simply cataloging a series of one or more short overlay programs in a special disc file (SM or USM). The first or root overlay is cataloged in this "system module" file under the name of the three-character keyword used to start it. Secondary overlays may be cataloged under three-character program names that have a $ as the second character. These overlays operate in MAP Ø in the privileged mode.

1.15.4 INPUT SPOOLER

The Input Spooler is an optional symbiont task that collects job streams from one or more job-entry devices for eventual use by one or more batch-processing tasks. The spooler places the job streams on secondary storage files while interpreting the job streams to determine the resources necessary to run them.

1.15.5 OUTPUT SPOOLER

The Output Spooler is an optional symbiont task that accepts data from what appears to be many imaginary printers and other hard-copy devices. The spooler moves all of this hard-copy information from the calling program to memory areas within the spooler -- and eventually to secondary storage files so that such imaginary devices appear to be unusually high-speed printers. As printed files are completed on the disc and as real printer(s) become available, the spooler moves the printed data from disc to printer while continuing to accept additional print data from program to disc. The task improves throughput by isolating the program from the slow device.

1.15.6 USER-CODED SYSTEM TASKS

The user is normally capable of executing all applications programs in the unprivileged state. MAX IV allows the user to add a privileged "system" task that allows program execution in the privileged state. Such a task must be viewed as any custom system element (for example, custom I/O handlers) in that it must be thoroughly debugged before its safety can be judged. Such system elements must, therefore, be installed when critical real-time processes are off-line. Such tasks can operate in their own addressing space or in MAP Ø with the system nucleus (if resident).

1.16 SYSTEM BATCH-PROCESSING PROGRAMS

All batch-processing programs execute as overlays of existing batch-processing tasks and are called explicitly by Job Control Directives.

The MAX IV Operating System has several standard batch-processing programs, sometimes referred to as "system" processors. Refer to Section 1.9.1Ø. These processors fall into two categories: Translators and Utilities. Each of these programs is documented fully in a separate manual; refer to the manuals listed in the Publications Catalog.

1.17 USER-CODED BATCH-PROCESSORS

Any suitable user-coded program may be executed as a batch-processor if its execution does not require critical real-time response. These programs may be used in several ways. If it is a production program that is used regularly, it should be completely link-edited and cataloged by the Program Manager in the system's batch-processing load module file. If execution of the program is infrequent, it may be compiled, assembled, and link-loaded directly in memory when it is time to execute it. This may take less time -- but more main memory -- because the program, its external linkage symbol table, and the link loader overlay program must exist in memory at the same time.

CHAPTER 2
HARDWARE REQUIREMENTS AND UTILIZATION

This chapter describes the hardware and memory requirements for the MAX IV Operating System.

2.1 DEVICE REQUIREMENTS AND USE

MAX IV requires a CLASSIC 78xy computer with:

- Memory mapping registers
- General register context file (16 blocks of 15 registers each)
- MAX IV Executive Features option

Basic hardware requirements are shown in Figure 2-1.

A terminal device (for example, teletypewriter or CRT) operating in a half-duplex mode is required as a logging device and command entry device for the system operator.

For each batch-processing task, a job-entry device (for example, a card reader or terminal) and a printing device are required. Spooling multiplies one physical printer into many "imaginary printers" but adds additional main memory requirements to the operating system. A spooler improves general system performance for these devices in multiple batch environments by keeping them busy and by separating the program's capabilities from the device's capabilities.

Other hard-copy devices (for example, card punch, paper tape reader/punch and floppy disc) are optional. These devices are recommended if no removable pack disc is included in the system.

2.2 MEMORY REQUIREMENTS AND USE

Memory requirements and use is discussed in the following sections:

- Main Memory
- Memory Expansion Limits
- Secondary Storage Requirements
- Hardware Memory Maps
- Actual Memory
- Virtual Memory
- Non-MAP Ø Resident Code

2.2.1 MAIN MEMORY

Memory requirements depend upon the user's application and the complexity of the MAX IV configuration chosen at system generation time. A typical system without the File Manager or MAXNET IV is estimated below:

	WORDS (K=1024)
MAX IV resident nucleus with system tasks X, OC, and ROL	35K
Exclusive Real-time task region	8K
User area shared by real-time tasks and one batch-processing task	14K
TOTAL	57K

The above configuration represents a practical system that a batch-processing task is "rolled-out" when any real-time programs need more memory than is currently being used by the batch-processing task. (Up to 14K words of real-time programs can execute before roll-out of lower priority real-time programs becomes necessary.)

A full MAX IV configuration with multiple batch-processing tasks, output spooling, File Manager and more "memory-independence" between batch-processing and real-time programs, takes additional memory.

	WORDS (K=1024)
MAX IV resident nucleus with X, FMX, OC, and ROL tasks	45K
Output Spooling (S) task	4K
Exclusive real-time task region	20K
Memory for multiple batch-processing tasks (concurrent)	28K
TOTAL	97K

This system can sustain a high degree of multiprogramming between several real-time and several batch-processing tasks before roll-in/roll-out becomes necessary.

2.2.2 MEMORY EXPANSION LIMITS

Currently, memory configurations with up to 4096K bytes of main memory are supported by MAX IV (a CLASSIC 78xy computer supports 4096K bytes of memory).

2.2.3 SECONDARY STORAGE REQUIREMENTS

MAX IV is a disc operating system and is not configurable unless a disc or its equivalent is available in the hardware configuration. (A CPU-to-CPU link may be programmed to "simulate" a disc for a remote computer, that utilizes a real disc in a host computer.) Thus, at least one disc is required for the minimum configuration.

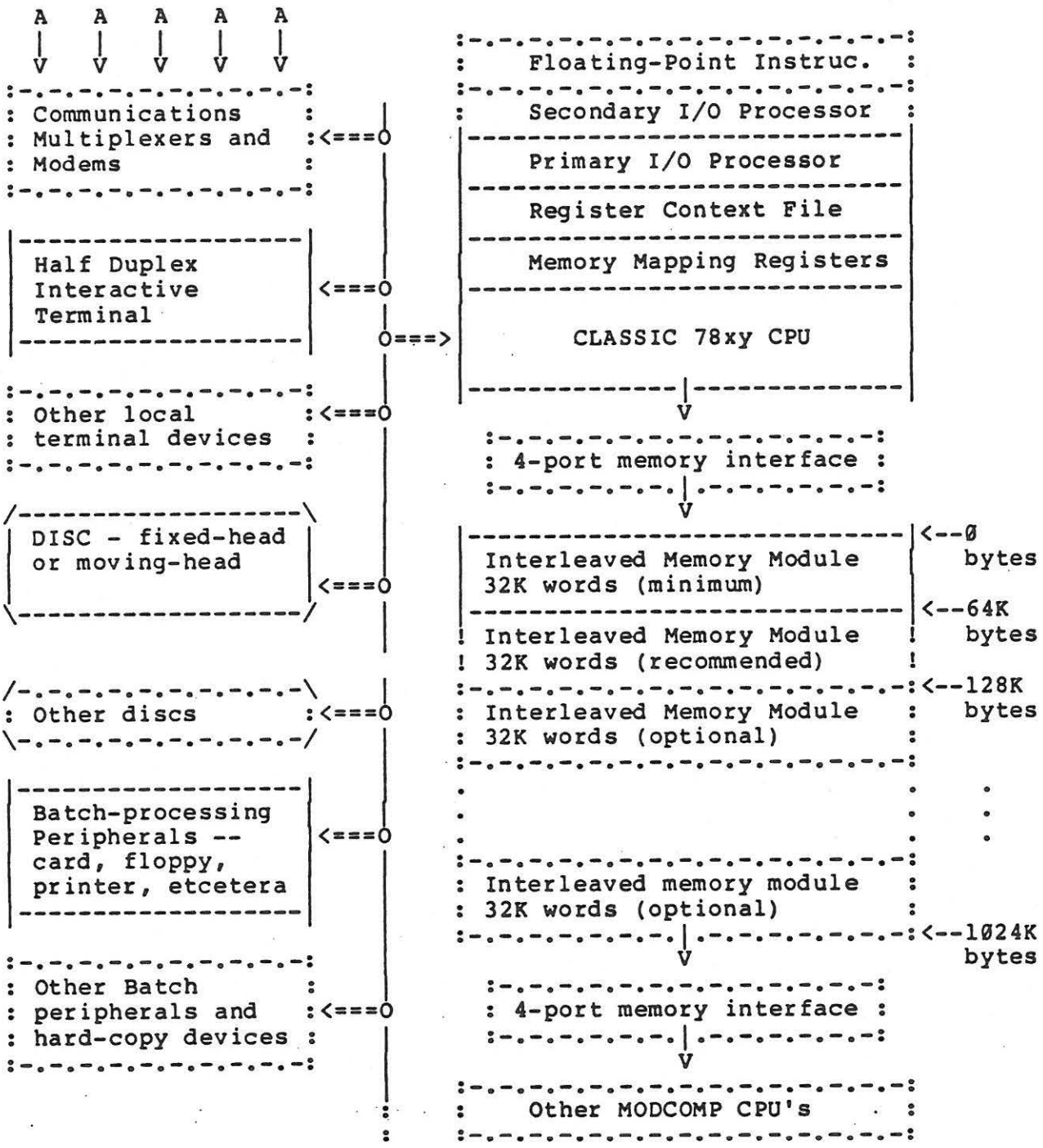


Figure 2-1. MAX IV Basic Hardware Requirements

Symbols used in Figure 2-1: (required) (optional)

Applications that plan to utilize multiple batch-processing tasks, large data bases, or a large degree of roll-in/roll-out activity should have at least two disc transports -- one should have removable media. If one removable pack can be dedicated for batch-processing use, a considerable amount of flexibility exists that allows the system data base to be changed online.

If roll-in/roll-out activity is to be the rule rather than the exception, a fixed-head disc is recommended for the system residence pack. In this environment, such a disc dramatically improves performance as opposed to a similar movable-head disc.

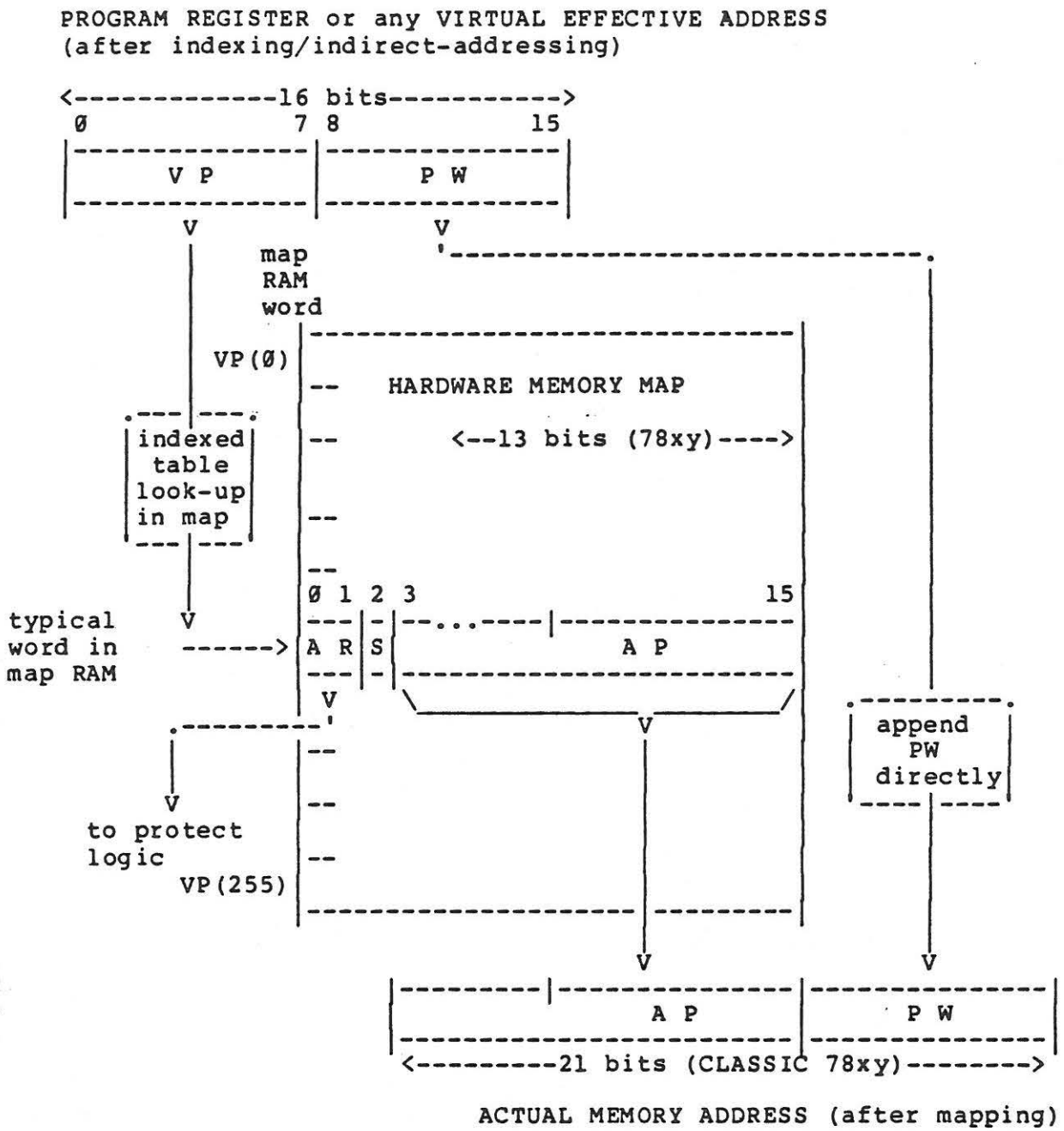
2.2.4 HARDWARE MEMORY MAPS

MAX IV uses the hardware memory maps of the CLASSIC 78xy computer to provide the hardware "relocation" capability of nonresident programs and to avoid memory fragmentation. These maps can be thought of as though they are separate 64K words memory areas. In the CLASSIC 78xy computer, one map can be subdivided into four 16K words memory areas for executing smaller programs. In this mode, the small maps are fully isolated from each other, and address references above 16K words are trapped.

Actually, these maps do not represent physical memory areas at all, but are switchable addressing ranges. These addressing ranges are sometimes called virtual addressing spaces. The large maps are each addressable from 0-64K words, and smaller maps are addressable from 0-16K words.

Actual memory is configurable to 8192 pages on a CLASSIC 78xy computer. Actual memory pages are not all necessarily accounted for in the maps. Some memory pages may even be duplicated in each map (for example, global shared memory areas). These maps are "memory address patch-boards" that allow any physical memory page, located anywhere in actual memory, to be addressed by any convenient 256-word range of a program's virtual address range.

A map is actually a high-speed "scratch-pad" memory (RAM) and may be modified or loaded quickly. MAX IV can manage these maps efficiently in a multi-programming environment, assigning them to tasks that are activated, and to tasks that interrupt others. User tasks use all the maps except one, while resident system tasks and all system services and traps use MAP 0. The 16-bit virtual memory addresses of all instructions are converted to 21-bit actual memory addresses as shown in Figure 2-2. The Program Register is also "virtual" and is converted in an identical manner for each instruction fetch.



LEGEND: VP = virtual page number
PW = page word number (offset)
AP = actual page number
AR = access right code
S = sharing control bit

Figure 2-2. Conversion of Virtual Addresses to Actual Addresses

2.2.5 ACTUAL MEMORY

A MAX IV operating system is composed of resident memory elements and nonresident (disc-resident) elements. The resident elements, called (collectively) the nucleus, permanently occupy the lowest pages of actual memory. This actual memory is composed of solid-state modules organized in 256-word pages. Extensive hardware instructions for memory allocation/deallocation, and the existence of memory maps, permit MAX IV to utilize any page anywhere it is needed; thus, multiple memory pools and fixed memory partition areas are not required, since they are "straight jackets" used to solve memory fragmentation problems and the lack of relocation hardware. Figure 2-3 illustrates how the actual memory of a CLASSIC 78xy computer looks when MAX IV is in use.

The fixed part of actual memory contains the elements that are permanently resident once MAX IV has been loaded and is operating. These elements are specified only by a system generation process and include:

- Dedicated cells for hardware interrupts, DMP (Direct Memory Processor) channels, MAP 0's map image, Memory Allocation Block (bit-map).
- The resident MAX IV operating system nucleus that includes REX service processors, I/O device handlers and interrupt drivers, schedulers, and other interrupt-driven elements.
- System tasks for error processing, spooling, roll-in/roll-out, and communications.
- Fixed buffers for nonresident services and directives.

The Transient Part of actual memory satisfies the transient requirements for memory during MAX IV operations. The elements in the transient part are scattered rather than contiguous. The CLASSIC 78xy memory maps organize the scattered actual memory pages into contiguous virtual addressing spaces. The scattering is a natural result of memory allocations/deallocations of various sizes being done at various, unsynchronized times during the course of running MAX IV. The following items are scattered throughout the transient part:

- File Manager FCB's (File Control Blocks)
- Nonresident user tasks and overlays
- Nonresident system tasks and overlays
- Batch processing tasks and overlays
- Private "inter-task" shared memory regions
- Transient I/O operation buffers and control block structures
- Task Control Blocks and work areas of nonresident tasks

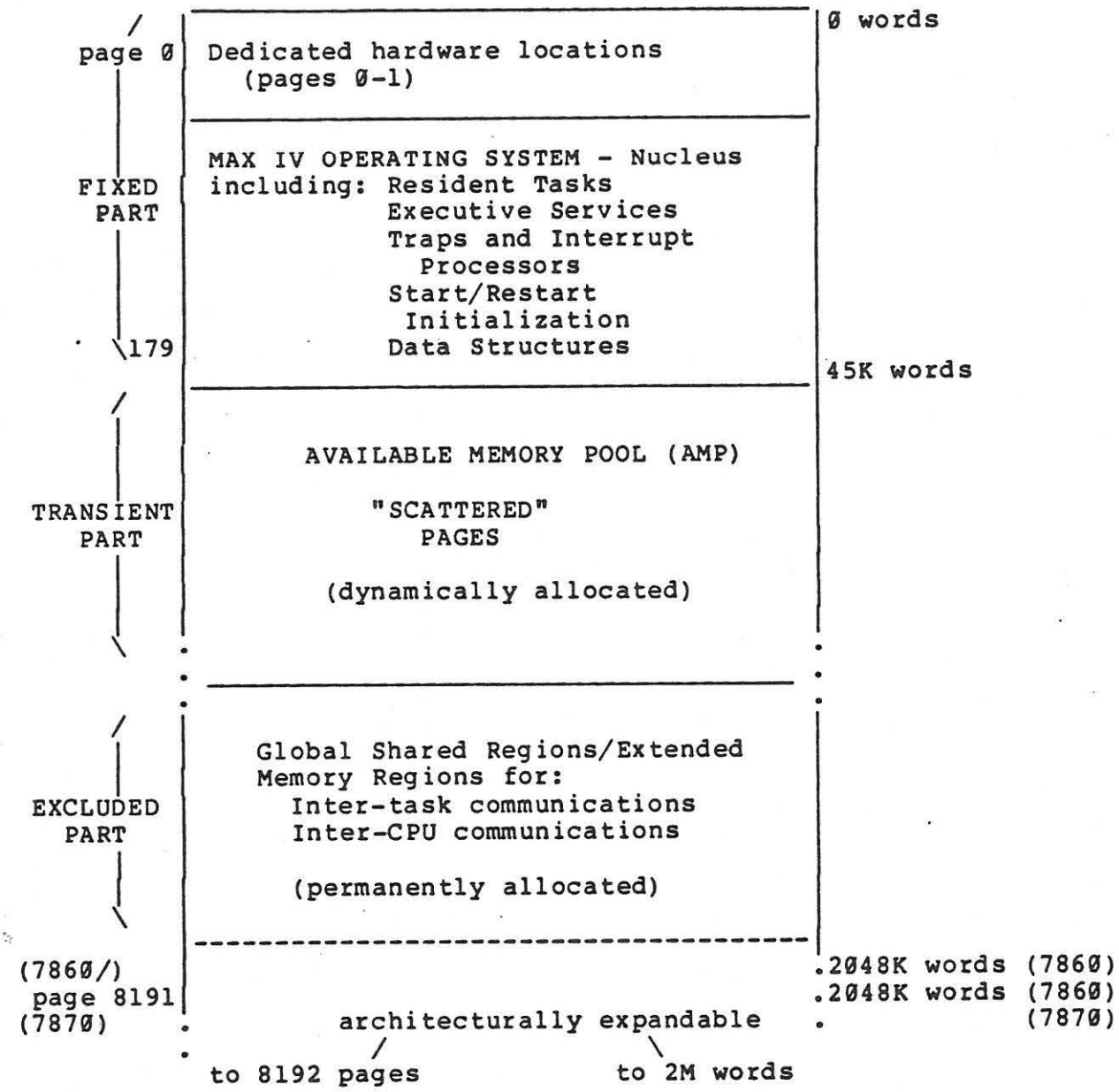


Figure 2-3. MAX IV Utilization of Actual Memory

NOTE: The 45K boundary between the nucleus and the AMP in Figure 2-3 is typical but can vary with operating system features chosen by the system generation.

The excluded part of actual memory contains contiguous regions that have been excluded from being available for use by MAX IV. Permanent global shared regions can be fixed in this part. These regions are addressable by any user task if specific Executive Services are called to insert the regions in the virtual addressing space of a task. These regions may be used as extended memory in a CLASSIC 78xy computer. Such regions may be fixed, at system generation time, at absolute actual page boundaries -- that may be configured as shared memory for more than one CPU.

2.2.6 VIRTUAL MEMORY

MAP Ø is dedicated to the MAX IV Operating System enabling it to address a full 64K words of memory. However, the operating system rarely, if ever, has 64K of actual memory allocated to the map for its use. MAX IV REX Services and MAX IV interrupt processing elements will always select this map automatically when called upon or when interrupts occur. MAX IV and any system task can execute privileged instructions when this map is selected.

2.2.6.1 MAP Ø - The Dedicated System Map

Figure 2-4 illustrates MAP Ø addressing space. Most of the map's contents are not changed once MAX IV is operating. These areas of virtual memory are actually the major fixed part of actual memory described above. All of this fixed part of actual memory is not necessarily included in the system map (is not accessible to its elements). Such areas include the global regions and shared load modules that do not need to be accessed by the system itself or by system tasks. System restart initialization and virtual initialization routines are also missing from the system map since they are not required after system start-up and are deallocated from virtual memory. No user tasks are in the system map, although special instructions allow MAX IV to access operands in such tasks whether or not they are currently executing. The transient elements of the system map include the Map Image and the TCB's of each transient user task. Leftover virtual memory address space is organized into a pool. Such space is granted only to tasks or REX services that execute within MAP Ø in the privileged state. Allocation of virtual space in this pool must be in one-page increments since contiguity cannot be guaranteed after initial system start-up.

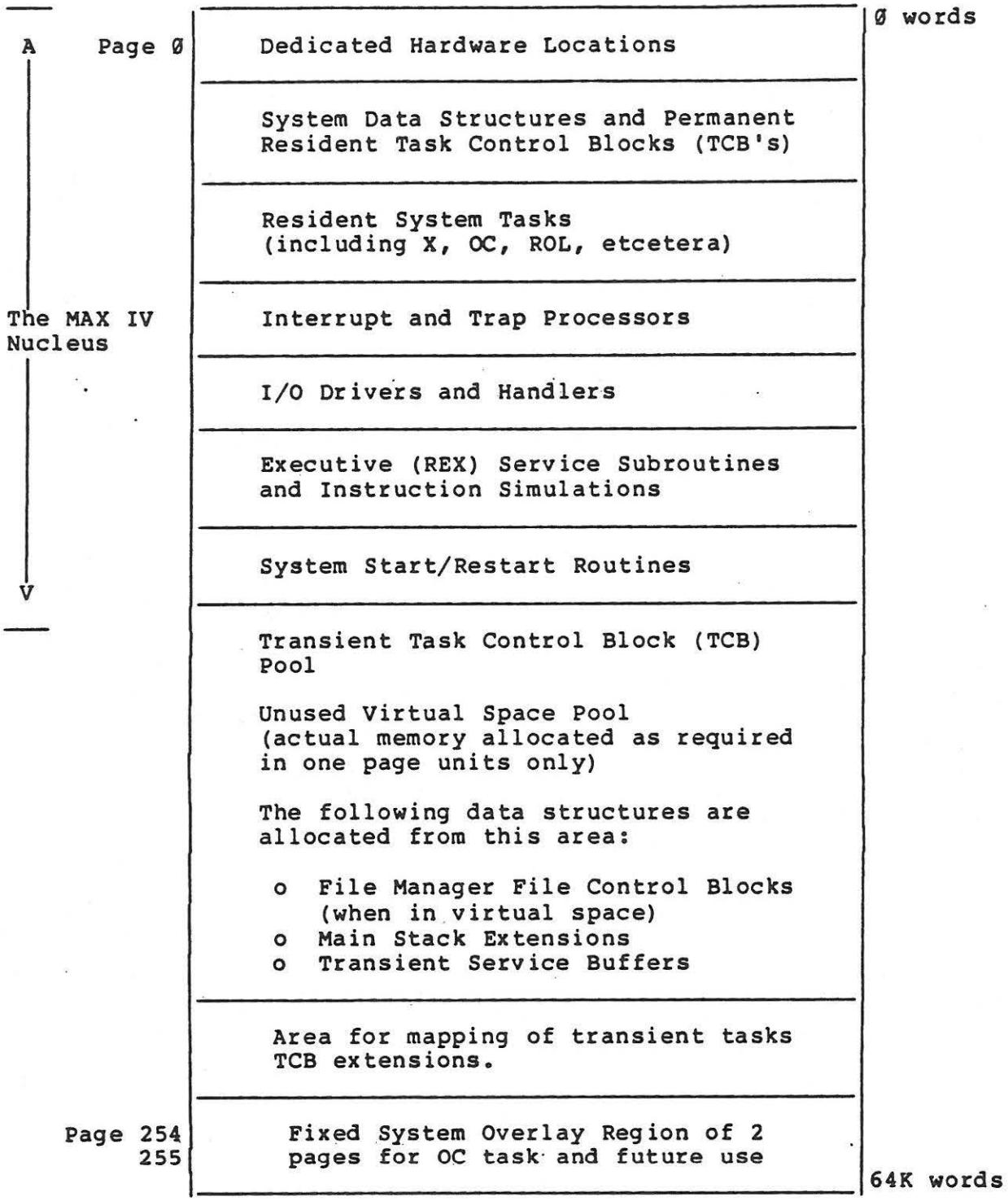

Ø words
A
Page Ø
Dedicated Hardware Locations
System Data Structures and Permanent Resident Task Control Blocks (TCB's)
Resident System Tasks (including X, OC, ROL, etcetera)
The MAX IV Nucleus
Interrupt and Trap Processors
I/O Drivers and Handlers
Executive (REX) Service Subroutines and Instruction Simulations
System Start/Restart Routines
V
Transient Task Control Block (TCB) Pool
Unused Virtual Space Pool (actual memory allocated as required in one page units only)
The following data structures are allocated from this area:
o File Manager File Control Blocks (when in virtual space)
o Main Stack Extensions
o Transient Service Buffers
Area for mapping of transient tasks TCB extensions.
Page 254
255
Fixed System Overlay Region of 2 pages for OC task and future use
64K words

Figure 2-4. MAP Ø Addressing Space

2.2.6.2 Maps For User Tasks

Memory MAPS 1, 2, and 3 are dynamically assigned to large tasks, and MAPS 4, 5, 6, and 7 are dynamically assigned to small tasks -- as tasks become active or interrupt each other. These tasks, or overlays, are normally positioned at the virtual memory address Ø of their assigned map by a linking process (see the MAX III/IV LINK EDITOR Programmer's Reference Manual listed in the Preface). Each such task treats a full 64K (or 16K) of virtual addressing space as if it were the only program in a computer of that size. Once a task's memory allocations have been loaded into a hardware map, it is probable that they remain until the task finishes execution. However, if six (or more) other tasks interrupt it during execution, its memory map may be reassigned while interrupted. When the interrupted task is resumed, the scheduler (TASKMASTER) retransmits its memory allocations to a hardware memory map made available to it at that time. A Map Image (MI) of each active task is contained in its Task Control Block extension so that the system can retransmit the task's memory allocation information to a hardware map when necessary. This Map Image is also used directly by the I/O handlers and DMP's (Direct Memory Processors) to assist them in passing data into and out of the actual memory assigned to a task, even though the task may not currently have its image contained in a hardware map.

Figure 2-5 illustrates a typical user task's virtual addressing space. MAPS 1, 2, or 3 are used in the example -- although MAPS 4, 5, 6, or 7 could be used if the task uses less than 16K words total. A task may actually change maps without being affected if it is interrupted. MAP 3 is not used when small tasks are assigned to the smaller 16K maps. The TASKMASTER also assigns small tasks to use MAPS 1 and 2 if they are the only maps available.

The Private Shared Load Module region is optional and may be used by the task to insert reentrant subroutines and other load modules that are to be shared with other tasks. The first task to request that such a module be inserted in its addressing space experiences the loading overhead. Subsequent tasks that insert the module use the copy of the module already in use by other tasks. If the last task using such a module exits the system, then the module also disappears and its memory is deallocated.

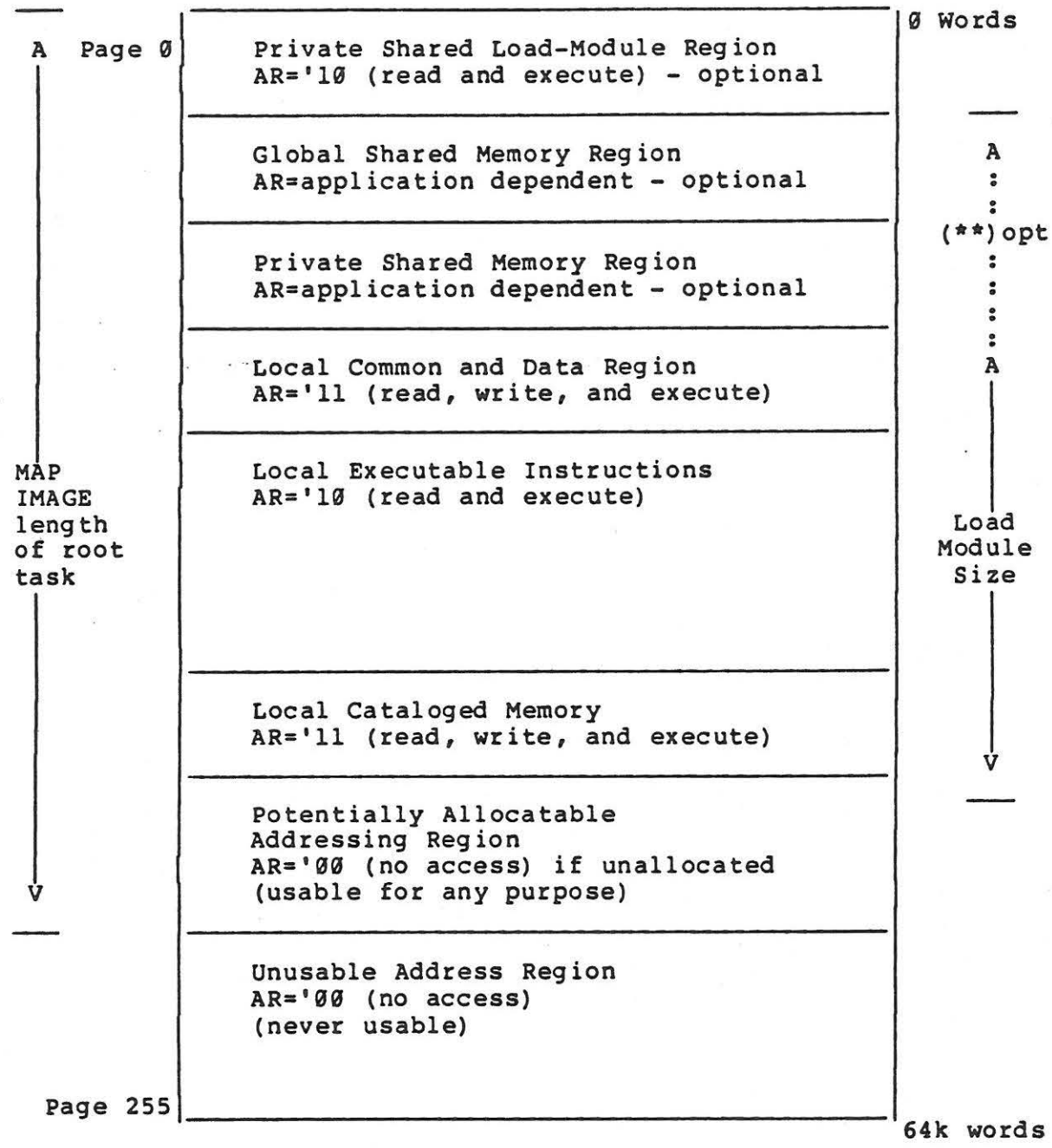


Figure 2-5. Addressing Space of a Typical User Task.

NOTE: A load module may load instructions or data into global or private shared memory regions if it can pass the security criteria established for such regions.

The Global Shared-Memory region contains one or more "public" and permanently allocated data areas that are fixed in actual memory at system generation time. These areas are sometimes called "global common" areas. The areas may be inserted and deleted from a program's addressing space (on page boundaries) at run time or they may be bound to the program when it is cataloged. The regions are used for inter-task communications. More information may be found in the "Memory Management Facilities," chapter of this manual.

The Private Shared-Memory region is optional and is an integral number of pages of virtual memory that may be used by a root task to communicate "privately" with one or more specific root tasks. Such regions are not as "public" as global regions and are actually allocated from the memory pool just as local private memory pages, except the "creating" task declares them "shared", which allows other tasks to insert them. These regions are deallocated when the last task using them exits the system. The creating task has full access to these regions -- the creating task can then set the access conditions for other tasks.

The Local Common and Data region is an area used by the program for its modifiable data buffers, arrays, or other variable information that must be changed during execution. The initial part of this area may be used by an overlay of the "root" task to leave information for access by subsequent overlay program segments. If a program makes this area an integral number of pages and isolates all its modifiable areas within it, then it is rewarded with a valuable debugging feature described below. The local common area is assigned full-access rights that permit read, write, and branch freedom within it.

The Local Executable Instruction area is self-explanatory. If this area and the common area are both constrained by the programmer to start on integral page boundaries and if these areas are made "pure" (that is, all executable instructions are separated from the modifiable memory elements) -- the user task has the executable instruction area set to an access right that permits freedom of read and branch, but prohibits memory write. This is valuable during task debugging, since only the write-accessible areas require dumps after program aborts. Any attempts by an undebugged program to change itself is trapped and identified immediately. This organization also aids the system in automatic checkpointing (roll-out) of the task should all or part of its allocated memory be required by a higher priority task.

The Local Cataloged Memory region is an area of memory specified for the program when it is cataloged by the TASK/OVERLAY CATALOGER. Many programs designed to operate in "fixed partitions" in other operating systems often expect to have such a "scratch" region assigned to them. These programs can determine the size of the region by calling an Executive Service. The region's size may also be changed at run time by the JOB CONTROL program (if the program runs as an overlay of a batch-processing task) or by calling other Executive Services. Full access rights are always assigned to the pages of such regions. A program may use these areas for transient I/O buffers or overlay extension areas that are not part of the root task body of executable instructions.

The Potentially Allocatable region is the number of pages of unallocated virtual memory remaining in the Map Image of the root task after the program is loaded. The task's Map Image can be sized when it is cataloged to allow for virtual memory usage beyond the "natural" size of the root task program. This region may be used to allocate local memory dynamically or to insert any type of shared memory region.

The Unusable Address regions of a program include those pages that are above the defined "Map Image" length of the root task. Such pages can never have memory allocated to them. These addressing regions, as well as any regions that do not currently have memory allocated or shared regions inserted, are given access rights that trap the task if they are accessed. For tasks that are assigned to small (16K) memory maps, this region also defines the addressing range above 16K.

2.2.6.3 Two Map Tasks

MAX IV language processing programs (such as FORTRAN, assemblers) provide facilities that permit a programmer to write programs in such a way that instructions are isolated from operands. Because the CLASSIC 78xy hardware (Program Status Doubleword) permits a unique map assignment for instructions and a unique map assignment for operands, a service for executing very large programs (up to 128K words of addressing space) exists. Such a program is assigned to two available memory maps. Indirect address constants are considered as operands, and machine instruction addresses are considered as instructions. The "immediate" instruction set may be used to communicate across these boundaries, because its operands are always in the instruction address space. Figure 2-6 illustrates how such a large program might appear to itself.

2.2.7 NON-MAP Ø RESIDENT CODE

Due to the increasing number of tasks simultaneously executing and the increasing number of MAP Ø resident optional features (MAXNET IV, File Manager, Event Logging, FORTRAN IV Resident Runtime, and Intertask Communications) there is a corresponding increase in the demand for MAP Ø virtual space. To decrease the number of optional features in a user's configuration or the number of tasks simultaneously in execution is not viable for all users. Some applications demand large numbers of tasks simultaneously in execution as well as one or more resident optional packages. The Non-MAP Ø Resident Code capability provides some additional relief by moving three of the optional packages out of MAP Ø. It will move any combination of:

- MAX IV File Manager
- FORTRAN IV Resident Routine
- Intertask Communication System

MAXNET IV may be SYSGENed to run largely as Non-MAP Ø Resident Code. Refer to the MAXNET IV System Installation Guide for details.

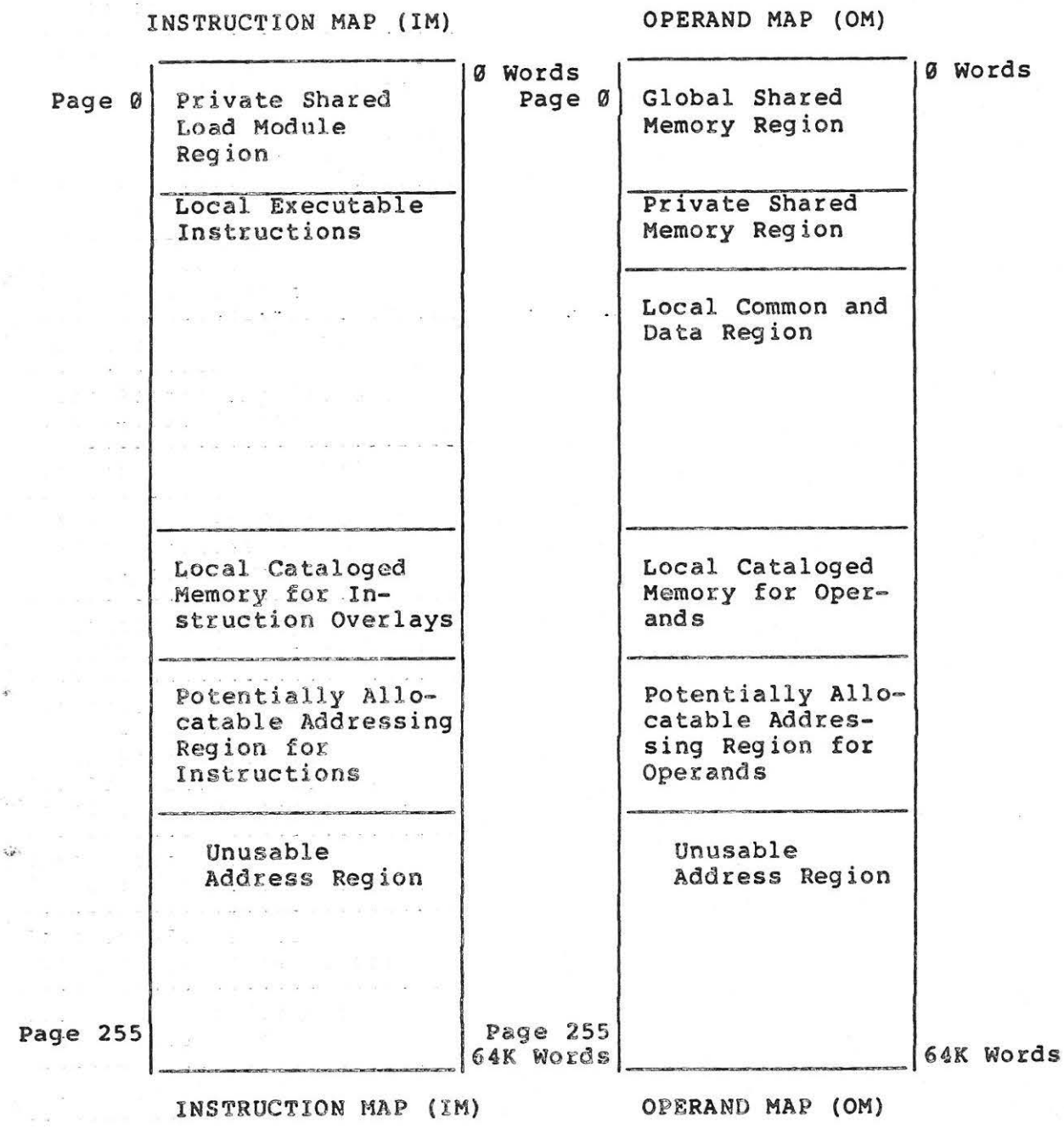


Figure 2-6. Addressing Space of a Typical Two Map Task.

At most, only two of the packages may be moved out of MAP Ø with one MP.TSK. If more than two options are selected, errors will be generated during assembly. To move more than two packages out of MAP Ø, create two MP.TSK's, giving a unique name to each.

The following steps are performed in order to map the optional packages into a map other than MAP Ø.

1. Assemble Non-MAP Ø Resident Code Mapping Task:

   a. Assemble MP.TSK from the system MC file:

      - Use the $FM option for the File Manager
      - Use the $FR option for FORTRAN Runtime
      - Use the $IT option for the Intertask Communications System

   b. Assemble MP.END from the system MC file:

      - Use the $FM option for the File Manager
      - Use the $FR option for FORTRAN Runtime
      - Use the $IT option for the Intertask Communication System

2. Link edit MP.TSK:

   a. Assign BI to the file containing the object of MP.TSK.

   b. Assign UL to the system object library (AL4):

      - The File Manager object must be on the UL file if $FM
      - The FORTRAN Runtime object must be on the UL file if $FR
      - The Intertask Communications object must be on the UL file if $IT

   c. Assign LB to the file containing the object of MP.END.

   d. Assign BO and SC files as usual.

   e. Link edit the objects using the MAX IV Link Editor (M4EDIT).

   f. The edit map must show the following ordering:

      (1) MP.TSK
      (2) FORTRAN Runtime, if configured
      (3) Intertask Communications, if configured
      (4) File Manager, if configured
      (5) MP.END

3. Catalog MP.TSK:

   a. It must be on a load module file directorized by the MAX IV Task Overlay Cataloger (TOC).

   b. It must be cataloged with the following resources:

      - o Priority of Ø
      - o Influence limit of Ø
      - o Privileged
      - o Autostart
      - o It must be a task

   c. If the LO option is on, informational messages are printed. (Error messages will always be printed.)

   d. If the Map option is on, the task will hold prior to unloading its map image.

4. Modify SYSGEN source:

   a. Add a PRESCHEDULE MP.TSK,,lmf,NOPERM for each MP.TSK.

      NOTE: If more than one map task is used, different names must be used.

   b. The MP.TSK PRESCHEDULE must be the first PRESCHEDULE statements.

   c. Taylor SYSGEN to your application as follows:

      - o If using FORTRAN IV Resident Runtime do the following:

        - Do NOT add a SERVICE FORR statement.
        - Remove FORTRAN IV Runtime for the LB file.
        - Re-link edit all FORTRAN IV programs.

      - o If using File Manager do the following:

        - Change RESSERVICES ALL to RESSERVICES MAPPED.
        - Recatalog all tasks which reference the File Manager to have a main stack size of 250 words.

      - o If using Intertask Communications do the following:

        - Specify MAPPED for parameter 3 on the Intertask statement.

   d. Assemble, Link Edit, and store the SYSGEN.

If you have difficulty moving the optional packages out of MAP Ø, check the following:

- o Assembly options used on MP.TSK and MP.END.
- o Ordering on link edit map.
- o SYSGEN source changes.
- o Load module file contents.

2.3 DISC LAYOUT

2.3.1 DISC CONTENTS

MAX IV uses a random-access storage device (disc) for elements of the system (and the user's application) that do not justify their permanent residence in main memory. There is no rule for determining which elements should be resident but, basically, any element with access time that is not critical (available in "milliseconds") and with an actual utilization that has a low duty cycle (less than "20%" of all computer time) is usually nonresident. Certain user-coded Executive services, operator directive processors, system tasks, and user tasks fall into this category.

Other operating system elements are stored on disc even though they remain resident in memory all or most of the time. The image of the resident MAX IV operating system nucleus is an example. Some system elements are made permanently resident even though their utilization duty cycle is low, because their instant access (when needed) is an important factor to system performance.

In all cases, disc residency is a convenience that provides for ease in updating or modifying elements in a state of development -- and is a way of extending the "dynamic" memory range of a system. A disc-resident task can be "established" when its utilization justifies being memory resident at all times or for extended periods -- but the ease of updating, because it is disc resident, is obvious. It may be "deestablished" when its residency is no longer required.

2.3.2 SYSTEM RESIDENCE PACK

Figure 2-7 illustrates a typical disc layout. The first pack transport (pack 0) is dedicated as the System Residence Pack. This pack may be either a fixed-head disc or moving-head disc. A fixed-head disc (or equivalent secondary storage device) is recommended where speed and efficiency is a concern.

The first part of the pack is dedicated for system use and consists of a series of fixed single-extent logical disc "devices" called "partitions". These partitions start on integral track boundaries. Some partitions contain special local directories that contain pointers to nonresident system elements. These files are maintained in an efficient format that does not require the overhead of the full file management system and its access methods. Special utility programs, which operate in either a stand-alone or batch-processing environment, are used to catalog and maintain these partitions or "low-overhead" files.

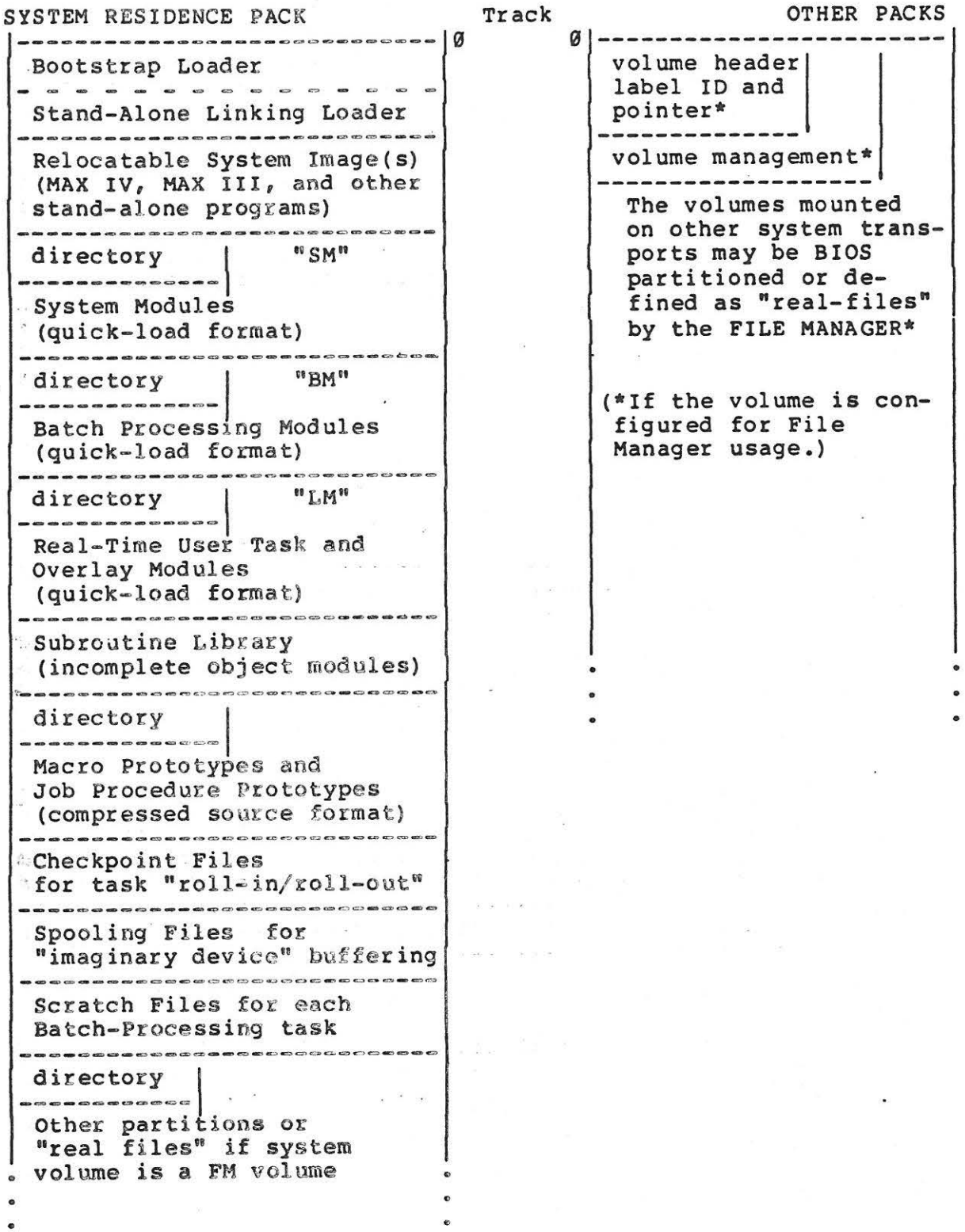


Figure 2-7. Typical Disc Layout

The first file is a two-track partition containing the Bootstrap Loader and the Stand-Alone Linking Loader. These programs are in a fillable format that allows them to be loaded with the standard CLASSIC computer's FILL switch. The Stand-Alone Linking Loader (SAL) is a program that computes the main memory size and then relocates itself into upper memory. Since it operates in the Absolute mode, it does not locate itself higher than 64K actual memory. Once loaded, SAL is capable of loading either MAX IV or other operating system images -- or any other stand-alone program (for example, diagnostics and other operating systems).

Other fillable programs may be installed on this first partition if it is enlarged. This partition is prepared by the System Disc Initialization Program that is available in a stand-alone fillable form on various media. When a system has more than one pack transport, a disc pack may be prepared online with a similar utility element of the FILE MANAGER.

The second partition is a sequential object module file containing the relocatable image of MAX IV (and other operating systems and other stand-alone programs and diagnostics). This partition is prepared and maintained by the MAX III/IV Library Update program (a batch-processing overlay). The programs contained on this file are only loaded into actual memory below 64K words by the Stand-Alone Linking Loader.

The next three partitions contain complete load modules of complete overlay programs or tasks. These files each have a directory that is maintained by the MAX IV Task/Overlay Cataloger program. All programs cataloged on these partitions are cataloged in a special "quick-load" or absolute format. These object modules are arranged in large multi-sector blocks on the disc medium. One or two disc accesses usually suffice in the loading of typical root tasks and overlays. Since user tasks are mapped in their own unique maps, always starting with the virtual address zero (Ø), these quick-load object programs do not need relocation, although they can still be designated relocatable if required for special applications.

Separate partitions and directories are used for the various system elements so that frequent recataloging and compressing of one partition does not "disturb" the more-or-less permanent system elements in another partition. Any of the load module files may option to have all or part of their directories reside in main memory. This capability reduces the number of disc accesses necessary to access a task or overlay.

The next partitions are called Subroutine Libraries. They are maintained by the MAX III/IV Library Update program and consist of a series of program elements in standard MODCOMP relocatable object module format. These program elements can be any incomplete (or complete) subprogram (or program) that may be selectively linked to any "main" program by the MAX III/IV Link Editor program (a batch processor). The FORTRAN Math Library, System Subroutine Library, System Generation Library, and other system object libraries reside in such partitions.

The next partitions contain information in compressed source format, which is cataloged and maintained by the MAX III/IV Source Editor program. Such partitions contain MACRO prototypes used for FORTRAN and other compilers, system generation, and executive (REX) service interface calls. Similar partitions contain the prototypes of Job Control procedures ($DO) for batch-processing tasks.

The remaining parts of the partitioned disc pack may be optionally allocated to various permanent scratch partitions used for various batch-processing, roll-in/roll-out, spooling, and other user-dependent functions. Some of these disc requirements may be made into "real files" as provided by the File Manager.

Any unpartitioned parts of the System Residence Pack (if it is a File Manager Volume) and any other disc packs may be initialized with special data structures that are managed by the MAX IV File Manager.

2.4 MULTIPLE MAP UTILIZATION

Because of the use of high-speed scratch-pad memories (solid-state RAM's) to implement the hardware memory maps, map images can be loaded or cleared very quickly when a task is activated or when the task interrupts a lower priority task and needs to use a map. However, because a reduction in task context switching time was one of the utmost goals of the MAX IV design efforts, methods were sought to reduce even further the small overhead imposed by having to reuse a single map for each task context switch operation.

In a typical priority-structured multiprogramming system, the average number of times a task interrupts a lower priority task is usually very low. With this in mind, all memory maps except MAP 0 are managed by MAX IV for nonresident task execution. The six highest priority tasks that are currently active are assigned to these maps. Only when the number of task interruptions exceeds 6 do memory maps need to be loaded during context switching operations. When resident tasks are also included in the application, they do not affect this context switching overhead, since they use the system's map, which is permanently assigned.

Memory maps never need to be SAVED when a task using one is interrupted, because a copy of each task's Map Image (MI) is maintained in memory as part of its Task Control Block (TCB). Memory map images need to be loaded into hardware maps only when an interrupted task resumes execution and its map was reassigned to a higher priority task while it was interrupted.

2.5 MULTIPLE GENERAL REGISTER BLOCK UTILIZATION

The CLASSIC 78xy computer makes 15 general-purpose, high-speed registers accessible to a programmer through most machine instructions. In addition, it has a 240-word hardware Context File (solid-state RAM) capable of saving 16 unique "copies" (or blocks) of these 15 general registers. At any one time a block of 15 words in the context file is assigned to "track" the contents of the

actual 15 high-speed Working registers whenever they are the destination of an operation. When hardware interrupts occur, any one of the context file copies can be automatically assigned to the interrupt level as a field in the Program Status Doubleword. When this occurs, the original working register block is effectively saved in the context file block that was "tracking" it, and the contents of the new register block is loaded into the working register block. This swapping of registers can take place in essentially the same time required for normal interrupt entry. This technique results in the following benefits.

- There are 16 blocks, of 15 general-purpose registers each, that can be used effectively by MAX IV.
- Interrupt routines can be executed without the time and memory overhead necessary for the program to save and restore the registers used by the routine.
- Interrupt levels can be assigned unique and dedicated general-purpose register blocks with contents that are restored automatically during interrupt subroutine entry and the contents survive from one interrupt to the next.
- Since all of the register blocks are not required for dedication to interrupt levels, the surplus blocks can be managed by the TASKMASTER (the system's task context switching routine) and assigned to different user tasks and system tasks. A typical MAX IV system uses four register blocks for interrupt levels and twelve blocks for assignment to the highest priority user and system tasks.

These characteristics and their benefits are discussed in detail in the following sections.

2.6 HARDWARE CONTEXT SWITCHING

Hardware context switching is a term used to describe the program switching action of a hardware interrupt. A program is interrupted, another process (the interrupt subroutine) is executed, and the original program is then resumed. The context switching time is usually only a measure of the entry time and exit time of the interrupt subroutine but must practically include the saving of all machine states (usually registers) that are necessary before the interrupt subroutine can do useful work, and the restoration of these states before the interrupt subroutine can return to the interrupted program. In a CLASSIC 78xy computer, the presence of the hardware context file simplifies interrupt subroutine execution. If an interrupt needs a whole block or part of a block of 15 general registers for its execution, a unique general register block (a section of the context file) is dedicated to the level. If only a few registers are needed by an interrupt subroutine, a general register block can be assigned that is shared with other interrupt levels -- by mutual consent. Not only are time and memory saved by not having to save the registers needed by the subroutine, but additional time is saved by letting many of its

most important pointers and data values remain in registers from one interrupt execution to the next. The additional time necessary to reload these arguments from memory on each interrupt is avoided, making these general register blocks similar to a high-speed memory extension.

Interrupt context switching also involves saving and restoring two additional CLASSIC 78xy registers -- the PR (Program Address Register) and the PS (Program Status Register). Together, they are referred to as the PSD (Program Status Doubleword). Four dedicated memory locations, organized into two pairs of words, are assigned to each unique hardware interrupt level for this purpose.

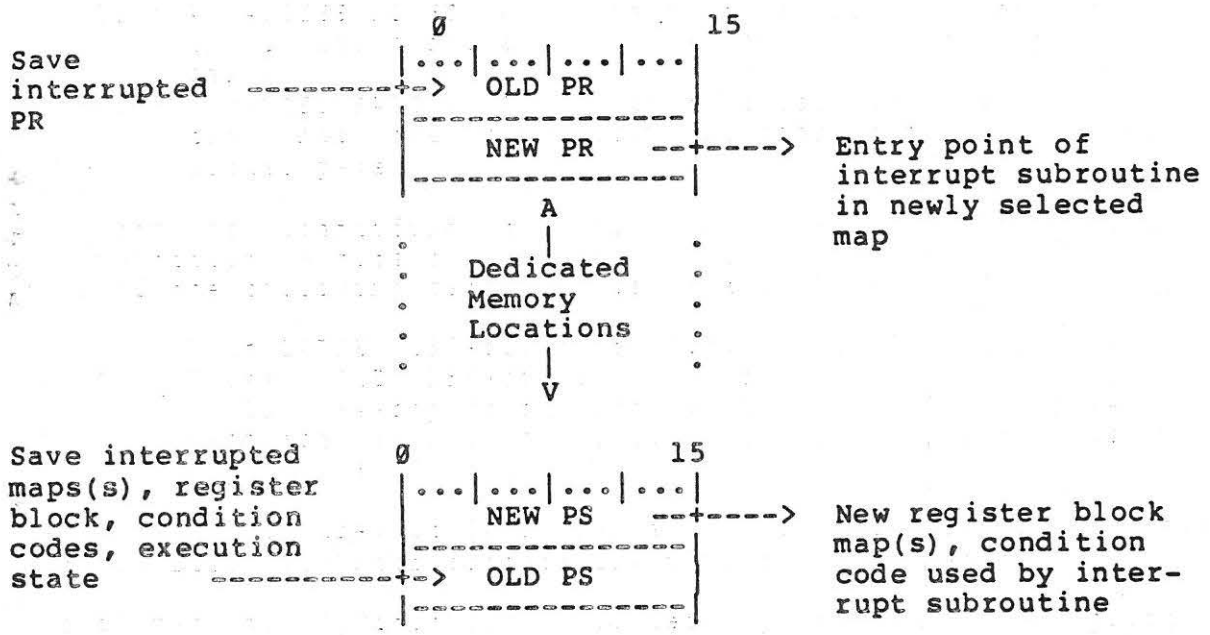


Figure 2-8. Hardware Context Switching

The NEW PS word of the interrupt has fields that select the states such as general register block, and memory map(s) to be used by the interrupt subroutine. When the general register block field in the OLD PS word (register file being used by interrupted program) differs from the general register block field assigned to the interrupt subroutine (NEW PS), the automatic register file swapping occurs. This feature does not penalize the user by wasting time, since it can take place in essentially the same time that is required to access the four memory cells necessary to effect the interrupt entry. This swapping process is reversed during interrupt exit -- but only two memory cycles are required. A complete hardware context switch can take place in only three microseconds.

MAX IV uses this capability to effectively reduce its own system overhead. Its most frequently used interrupt levels (TASKMASTER, Clock Interrupt, Data Interrupt Party Line, Service Interrupt Party Line) are assigned complete or partial dedicated register blocks. The most important pointers, addresses, and constants used by these subroutines are maintained in the dedicated registers and may not even exist in memory (except at system start). Other dedicated register blocks may be shared by the remaining interrupt levels.

Since the swapping of the PS words upon interrupt entry and exit also affects the assignment of memory maps (one for instructions and one for operands), an interrupt routine may automatically be made to use a different memory map than the program it interrupted. For most hardware levels, this new map is always selected to MAP Ø. Thus, all interrupt subroutines and their dedicated memory locations are mapped into the addressing space of the system map (MAP Ø). When an interrupted program is resumed, its memory map(s) is (are) automatically reselected.

2.7 SOFTWARE (TASK) CONTEXT SWITCHING

All user tasks and system tasks operate at "software-generated" interrupt levels created by the TASKMASTER routine that operates at the lowest hardware interrupt level (#F). Since this interrupt is the only level that always "knows" that it is interrupting the hardware background (non-interrupt) time of the CPU, it can effectively multiplex this hardware background time into many additional pseudo-levels, called task levels. The TASKMASTER has its own dedicated general register block in which it maintains the most important pointers (current TCB pointer, masks, pool pointers for available register blocks or Memory Maps) used for software context switching. The program it interrupts is always the currently executing task. The program it resumes is either a return to the original interrupted task or to another task that it chooses to run instead of returning to the interrupted program. Since the TASKMASTER can manipulate the "old PS" word of the interrupted program, it can assign surplus register blocks (those register blocks not dedicated to hardware interrupt levels) and surplus memory maps to any task when it transfers control to it (TASKMASTER interrupt exit).

If the number of tasks in the system is less than the number of available register blocks, software context switching can be accomplished without any general register files having to be saved and restored by the TASKMASTER.

In more complex applications, where the number of concurrently active tasks is greater than the number of surplus general register blocks, the TASKMASTER must save and restore registers when (or if) required. Even in these system applications, however, the average number of nesting levels is usually such that register saving and restoring is rare. Thus, another major contribution to operating system overhead can be reduced.

Usually 12 blocks of surplus general registers exist for assignment by the TASKMASTER. As tasks become active, each is assigned its own unique register block. As tasks become active (user tasks or system tasks), the needed register blocks are taken from the lower priority interrupted tasks. The register block contents of such tasks are saved in memory before they are reassigned to the higher priority tasks. If any task exits (terminates execution as opposed to simply relinquishing time), its register block is returned to a pool so that it can be assigned quickly to new tasks as they become active. If an interrupted task is reentered and its register block has not been reassigned while interrupted, then it is simply "switched on" by the normal restoration of its PSD instead of having to be restored from memory.

If only one general register block is used by the TASKMASTER, this single block has to be saved in memory (in the interrupted task's Task Control Block (TCB)) and restored from memory for each task interruption.

CHAPTER 3
MEMORY MANAGEMENT FACILITIES

This chapter discusses MAX IV memory management facilities. The following sections describe:

- The CLASSIC hardware facilities that provide the context for memory management in the MAX IV Operating System.
- Definition of terms used in the context of MAX IV applicable to memory management.
- Memory management operations.
- Organization of tasks for memory management in a MAX IV environment.
- Memory management services provided by MAX IV.
- System considerations in memory management.
- The ROLLER.

Throughout the remainder of this chapter, terms that are used in a precise meaning with respect to memory management are capitalized on the first appearance.

3.1 CLASSIC HARDWARE FACILITIES

3.1.1 MEMORY ORGANIZATION

The basic increment of main memory storage in the CLASSIC system consists of an ACTUAL PAGE of 256 16-bit words. A CLASSIC computer has a maximum of 131,Ø72 actual pages yielding a maximum main memory capacity of 2 megawords 16-bit words (4 megabytes).

3.1.2 MEMORY ADDRESSING

The location of a single word in memory is specified by a 21-bit ACTUAL WORD ADDRESS. The high-order 13 bits of this address constitute an Actual Page number (Ø-8191) and the low-order 8 bits contain the PAGE WORD number (Ø-255).

3.1.3 MEMORY ADDRESSING MODES

When the CLASSIC computer is operating in the Absolute mode, all memory addressing is carried out in ACTUAL mode. In this mode, any address that can be generated references a word in one of the actual pages numbered Ø through 255. (A reference to address 16384 (#4ØØØ) would access Page Word number Ø of Actual Page 64.) In this mode, the highest address that can be referenced is limited to that which can be expressed in 16 bits (or 65,535 = #FFFF).

In the CLASSIC computer's normal mode of operation, memory addresses are translated to permit any actual page that is attached to the system to be referenced. This mode of operation is termed the VIRTUAL mode and is the only mode of operation supported by the MAX IV Operating System. (While the effect of entering the virtual mode of operation is not solely limited to memory addressing, the other effects are ignored for the purposes of this discussion.)

The following paragraphs describe those portions of the CLASSIC hardware that effect the translation of addresses.

3.1.3.1 Mapping Register Files

Associated with the task when it is executing is at least one, and at most two, MAPPING REGISTER FILES (MAPS). A single map consists of either 64 or 256 hardware registers. A single register contains the following information:

1. An actual page number or "Ø".
2. An ACCESS RIGHT (AR) code that specifies what type of usage is permitted of the actual page by the executing task.
3. A SHARED-PAGE INDICATOR that describes whether another task has the capability of concurrently accessing the actual page.

Each map is identified by its MAP NUMBER. The hardware implementation permits concurrent, simultaneous usage of one or the other of the following sets of maps on the CLASSIC 78xy computer.

- Four LARGE (256-register) maps (Ø, 1, 2, and 3).
- Three LARGE (256-register) maps (Ø, 1, and 2) and four SMALL (64-register) maps (4, 5, 6, and 7). These SMALL maps are obtained by dividing LARGE map 3 into four SMALL maps (4, 5, 6, and 7).

When a map is in use, it describes a unique set of actual pages that are fully isolated from those described by other maps.

Because maps are dynamically assigned by the MAX IV Operating System, the user is normally not informed of which map the task is using.

3.1.3.2 Program Status Doubleword

Associated with an executing task is a Program Status Doubleword. This doubleword contains two 3-bit fields that specify which map is to be used for instruction fetching and which map is to be used for operand references.

The map used by the task for instruction fetching is called the INSTRUCTION MAP (IM), and the map used for operand addressing is called the OPERAND MAP (OM). All words of multiword instructions are fetched from the addressing space defined by IM. All operands and indirect address constants are fetched from the addressing space defined by OM.

Regardless of whether or not the task actually uses one or two maps when executing, the contents of these fields are always used to select a map for the translation process. (If the task uses only one map, both fields contain the same map number.)

3.1.4 ACCESS RIGHTS

The Access Rights (AR's) that are associated with each page of virtual memory determine the permitted degree of access to (or potential use of) that page.

The AR's associated with a page are specified by a pair of bits termed the "AR Code" with potential settings and meanings that are as follows:

AR Code	Degree of Access Permitted/Potential Usage
'11	May be accessed for operand read and write or for instruction fetch. Page may be used as operand or instruction area.
'10	May be accessed for operand read or instruction fetch, but operands may not be written. Page may be used for read-only operand and storage or program instruction storage.
'01	Operand read only.
'00	No access in any form. (No page of actual memory is associated with the virtual page addressed by the program.)

Table 3-1. Access Rights

The AR Code of a page is checked by all elements of the CLASSIC 78xy hardware system whenever the page is referenced. If the AR Code does not permit the usage requested, the access is not permitted and a System Protect trap (level two) occurs.

3.1.5 SHARED-PAGE INDICATOR

Associated with each page of virtual memory is a Shared-Page Indicator Bit. The setting of this bit is inspected by the Deallocation Service and if set, the deallocation of the actual page specified is inhibited. The bit is also tested by the Roller and if the bit is set the page is not rolled out.

This indicator may be set and reset only through use of software services provided by the MAX IV Operating System.

3.1.6 THE VIRTUAL MEMORY ACCESS PROCEDURE

As the task executes and memory is referenced, the mapping hardware translates the addresses generated as follows:

1. The memory address generated is separated into two fields:

   - o Bits Ø-7: the VIRTUAL PAGE (VP) number
   - o Bits 8-15: a Page Word (PW) number

2. Depending on whether an instruction or operand is being referenced, the IM or OM field contents of the Program Status Doubleword are used to select a map.

3. The Virtual Page number is used to select a register in the referenced map.

4. The AR code contained in the selected location is examined and:

   - o If the AR code indicates no access is permitted, a System Protect trap occurs at Interrupt Level 2.
   - o If the AR code indicates read-only access is permitted and an instruction reference is being made, a System Protect trap occurs.
   - o If the AR code indicates either read-only or read/execute access is permitted and operand storage is required, a System Protect trap occurs.
   - o Otherwise, the memory reference will be permitted and the translation process continues.

5. The Actual Page number contained in the selected register is used to select the actual page of memory referenced, and the Page Word number extracted at the first step is used to select the specific word of the actual page.

3.2 MAX IV MEMORY MANAGEMENT DEFINITIONS

The following sections define terms used in the MAX IV system with respect to memory management. The terms are defined here as a prelude to the discussions contained in the remainder of this chapter.

TERM	DEFINITION
VIRTUAL MEMORY	Virtual memory is memory that is addressed when the CLASSIC computer is operating in virtual mode. Since actual memory may or may not be associated with certain pages of this virtual memory, it may be helpful to think of it as one or more virtual addressing spaces.
MAP ZERO	MAP ZERO (or MAP Ø) is the term used to describe the virtual memory or addressing space occupied and used by the MAX IV Operating System. The term is derived from the fact that MAX IV permanently reserves MAP Ø for its use. The virtual memory defined by this map includes actual pages that are: 1. Hardware-dedicated for specific purposes. 2. Occupied by: o The permanent instructions and data structures of the MAX IV Operating System (the SYSTEM NUCLEUS). o Tasks that are installed at system generation to be resident in the map at system start-up. o The transient programs and data structures of the MAX IV Operating System.
ALLOCATED MAP	A map that has been assigned to a task for its use in execution is called an ALLOCATED MAP.

Table 3-2. Memory Management Definitions

UNALLOCATED MAP	An UNALLOCATED MAP is a map that is not currently assigned to any task. Immediately following start-up of the MAX IV Operating System, the set of maps available for allocation consists of the maps numbered 1, 2, 4, 5, 6, and 7. In some configurations of MAX IV, Map 3 is available instead of maps 4, 5, 6, 7.
MAP IMAGE	A MAP IMAGE is a data structure associated with a task. The data structure is resident in MAP Ø and consists of "n" words ("n" may take any value from 1 through 256). The task does not address this structure directly. The map image contents always reflect the current state of the virtual and actual memory associated with an addressing space of the task and are used: 1. To load locations Ø through "n-1" of an allocated map. 2. To determine actual memory addresses during I/O operations. 3. As both a control mechanism for and input to memory management services of the MAX IV Operating System
MAP IMAGE LENGTH	MAP IMAGE LENGTH describes the number of words required to contain a map image and may assume any value from 1 through 256. The length of a task's map image is determined when it is cataloged -- and determines the amount of memory that can ever be addressed by the task.
LARGE MAP	A map that is capable of containing the description of a virtual memory addressing space consisting of up to 256 actual pages is a LARGE MAP. In the CLASSIC 78xy computer there are three large maps initially unallocated after MAX IV Operating System start-up -- Maps 1, 2, and 3. Map 3 is not available for use in some configurations of MAX IV, or when available, Map 3 may not be used coincidentally with the small maps 4, 5, 6, and 7.

Table 3-2. Memory Management Definitions (Continued)

SMALL MAP	A map that is capable of containing the description of a virtual memory addressing space consisting of 1 to 64 actual pages is a SMALL MAP. There are four small maps initially unallocated after MAX IV Operating System start-up -- Maps 4, 5, 6, and 7. These small maps are not available for use in some configurations of MAX IV. These maps share hardware with large Map 3 and are unavailable for use when Map 3 is in use.
ONE-MAP TASK	A ONE-MAP TASK is a task that uses a single map as both IM and OM. Such a task is said to occupy a single addressing space.
TWO-MAP TASK	A TWO-MAP TASK is a task that uses two discrete maps as IM and OM. Such a task is said to be "split" since its instructions and operands are divided among two addressing spaces.
UNALLOCATED PAGE	An UNALLOCATED PAGE is an actual page that is neither assigned to any task as a component of the task's virtual memory nor otherwise reserved for some permanent usage in the system. All unallocated pages are said to be in the Available Memory Pool (AMP).
LOCAL PAGE	A LOCAL PAGE is an actual page that has been assigned to a specific task as a component of the task's virtual memory. A local page is solely accessible by that task.
SHARED PAGE	A SHARED PAGE is an actual page that may be a component of the virtual memory of one or more tasks concurrently.
GLOBAL SHARED PAGE	A GLOBAL SHARED PAGE is a shared page that is permanently reserved and defined for usage as a shared page. Actual pages that are accessible by two or more hardware systems are always defined as this type of page.
PRIVATE SHARED PAGE	A PRIVATE SHARED PAGE is a local page that has been redefined for usage as a shared page by a task. An actual page can belong to this class only as long as at least one task sharing the page preserves its definition as a shared page.
VIRTUAL PAGE	A VIRTUAL PAGE is an area of 256 words of virtual memory with an address in virtual memory that contains "Ø" in the low-order 8 bits of the 16-bit virtual address.

Table 3-2. Memory Management Definitions (Continued)

VIRTUAL REGION	A VIRTUAL REGION is an area of virtual memory within a single addressing space (and its associated map and map image) that is identified by specifying a Virtual Page number and the number of virtual pages in the area. By definition, a virtual region contains at least 1 and at most 256 virtual pages. As a system convention, the virtual region specified as beginning at Virtual Page Ø and containing 256 virtual pages always means "the entire virtual memory defined by the map (and map image)".
FREE REGION	A FREE REGION is a virtual region with which, at some point in time, no actual pages are associated. Accessing such a region causes a System Protect trap.
CLEAR AREA	The CLEAR AREA is that portion of potentially accessible virtual memory or addressing space that a task does not intend to access or use. The clear area in a map may consist of either: o Ø to 63 virtual pages in a small map. o Ø to 255 virtual pages in a large map. For any addressing space used by a task, the clear area may be defined to consist of those virtual pages with a Virtual Page Number that is greater than or equal to the associated map image length. Accessing the clear area causes a System Protect trap.

Table 3-2. Memory Management Definitions (Continued)

3.3 MEMORY MANAGEMENT OPERATIONS

All of the memory management services available use the map image(s) of a task's addressing space as the primary control structure for operations requested by or on behalf of the task. The normal operation of these services results in a modification of the affected map image followed by modification of the affected portion of any active map of the task.

Elements of the MAX IV Operating System that support the execution and operations of the task (the Module Loader primarily) also use these services at times when the task may not be assigned active maps (as during task loading, for example). The only effect of usage of these services at such times is to modify the map image. However, prior to permitting execution control to pass to the task, the latest contents of the map image are always loaded into the active maps of the task.

The following paragraphs discuss the types of memory management operations available for use in designing and implementing tasks.

3.3.1 LOCAL PAGE ACQUISITION/DISPOSAL

A task can acquire and dispose of actual pages as necessary during its residence in the system. The acquisition process is called "allocation" and the disposition process is called "deallocation."

Allocation is used as necessary by the Module Loader to load the task and any overlays called. When the task exits from the system, deallocation is used to free the task's local pages for re-use by other tasks. Hence, neither of these processes are ever explicitly invoked.

However, if a requirement to acquire additional memory for transient usage during the execution of a task exists, the allocation process may be invoked explicitly. In consideration of other tasks executing in the system and in the interests of over-all system operating efficiency, it is desirable to invoke the deallocation process when actual pages are no longer required by the task. This permits the pages to be made available for use by other tasks.

3.3.2 PAGE SHARING

Page sharing is a technique implemented in the MAX IV Operating System to permit more than one task to concurrently access one or more actual pages. The technique is accomplished by recording the address of the actual page(s) involved in the map image of one or more tasks simultaneously.

Using such techniques, several tasks may have access to a common set of operands or subprograms with an over-all savings in the amount of actual memory required to support concurrent operations of the tasks and/or increased operating efficiency. Within a particular CPU's local memory, instructions or operands may be shared. In systems with memory shared be tween more than one CPU, only operands may be shared. Refer to the section in this chapter entitled CLASSIC 78xy Shared Multiport Memory. A flexible Inter-task Communication Service is also provided by these techniques (refer to Chapter 7).

A shared-page region consists of a named set of one or more actual pages that are defined in one of two system directories and are accessible, on a controlled basis, to more than one task.

3.3.2.1 Shared Page Definition

Shared-page regions may be defined at system generation or by executing tasks. When such a region is created, the following information is entered in the appropriate system directory:

- Region identifier (name).
- Definition of the actual pages that constitute the region.
- Constraints upon usage of the region by tasks attempting to access the region.

There are no contraints on a task if the task is the creator of a region. However, other tasks' access to the region can be limited. To permit such limitation, the following mechanisms are provided:

- A "Read Lock" of 1 to 3 CAN code characters may be specified. If specified, a matching "Read Key" must be supplied by any task that attempts to read operands or execute instructions contained in the shared page region.
- A "Write Lock" of 1 to 3 CAN code characters may be specified. If specified, a matching "Write Key" must be supplied by any task that attempts to write operands into the region.
- The region may be classified as "Read Only" in which case -- regardless of the setting of the Write Lock -- no task is permitted to write operands into the region.
- The region may be classified as being accessible to privileged tasks only. In this case, no task operating in the unprivileged state is allowed to access the region.

Three types of shared-page regions may be defined in the MAX IV Operating System:

- GLOBAL SHARED PAGE REGIONS that are defined in the system generation process and are permanently reserved for use each time the system is started or restarted. The definition of such regions is independent of any particular task's residency in the system. The contents of actual pages constituting the region may consist of instructions or operands, but these areas are not initialized by the Operating System. These regions consist of contiguous actual pages if they are more than one page in length. Extended Memory Regions start on 8K word boundaries and extend for an integral multiple of 8K words.

- PRIVATE SHARED-PAGE REGIONS are defined by an executing task and have a definition that survives only as long as all tasks currently sharing the region have not passed through the EXIT service or otherwise ended accessing of the region. The contents of actual pages constituting the region are known only to the tasks sharing the region. These regions are contiguous in the virtual memory of the accessing task(s) but do not necessarily consist of contiguous actual pages. Such regions originate as local pages in the task that create them.

- SHARED LOAD MODULE REGIONS, defined by an executing task, have a definition that survives only as long as at least one task sharing the region has not passed through the EXIT service or otherwise ended its access of the region. The contents of a region of this type are defined by loading a load module into the region when the first task creates the region. The load module may be resident in either a global or a private file or the invoking task.

3.3.2.2 Shared Page Usage

The process by which a task obtains access to a shared-page region is called "insertion", and the process by which a task ends its access to such regions is termed "extraction".

Insertion is used by the Module Loader in the process of loading a task or an overlay called by a task. The process may also be explicitly invoked by an executing program of a task.

When a task initially attempts to access a shared (Non-Extended Memory) region, the access limitation mechanisms are applied as follows:

- If a task is unprivileged and only privileged tasks may share the region, the task will not be permitted to access the region.

- If a Read Lock exists for the region, the Read Key supplied is compared. Depending upon the results, the task is marked as having or not having the capability to gain "read-only" and "read/execute" use of the region.

- If the region is restricted to "read only" usage, the task is marked as not having the capability of writing into the area. No Write Locks are examined.

- If a Write Lock exists, the Write Key supplied is compared to it. Depending upon the results, the task is marked as having, or not having, the capability to gain "read/write" use of the region.

- Extended Memory Regions always have Read/Write access rights.

Since the first intended usage of the region must be specified when the region is inserted, access to the region depends upon the results of applying the mechanisms above.

Extraction may be explicitly invoked by an executing program. If the task exits from the system, any shared region that the task was accessing, but has not extracted, is automatically extracted by the EXIT service.

3.3.3 ACCESS RIGHTS MODIFICATION

A task may modify the degree of access to local pages (except setting an AR Code of '00) by invoking the appropriate system service during execution.

A task may also modify the degree of access to shared pages, provided that the access limitations of shared region usage do not restrict access.

3.4 TASK ORGANIZATION FOR MEMORY MANAGEMENT

In creating a task, values can be specified for parameters that are used to establish a context for memory management within a task or on behalf of a task.

All of the values specified at task creation are recorded permanently in the task's load module and are used -- each time the task is loaded into the system -- to establish the task's memory management environment in the Task Control Block.

The following paragraphs discuss each of the items of information that can be specified and also discuss the extent to which the choice of a value for an item affects the task's memory management operations.

3.4.1 NUMBER OF MAPS

Whether a task uses one or two maps during execution is specified when it is cataloged.

If one map is chosen, the task is limited to using a maximum of 256 virtual pages as both instruction and operand areas at any given point in execution.

If two maps are chosen, the task is limited to using a maximum of 256 virtual pages for instruction storage and a maximum of 256 virtual pages as operand areas at any given point in execution.

3.4.2 VIRTUAL ADDRESSING RANGE

For each addressing space used by a task, the option exists of specifying the VIRTUAL ADDRESSING RANGE to be used in the map. The virtual addressing range used in a space consisting of the addresses from Ø to "R" ("R" is a virtual address in the range 1 to 65,535 (64K-1)).

From the range specified, the system derives a map image length by rounding, if necessary, the supplied address up to the next integral virtual page boundary and using the Virtual Page Number obtained to determine the required map image length.

Once the virtual addressing range for an addressing space is specified, the task is limited in its operations in that space of the specified range. The specification is also used to determine:

- Whether a small or large map is required to define the virtual memory associated with the map during active execution by the task.
- The size of the clear area associated with the addressing space.

For instance, if it is specified that a task's OM uses the virtual addressing range from Ø to 16380 (=#3FFC):

- The length of the OM image will be 64 words.
- A small map is sufficient during execution.
- The clear area of the addressing space consists of 192 pages.

Specification of a virtual addressing range in an addressing space does no more than establish an environment for usage of virtual memory in the space. Such specification does not necessarily imply that every virtual address in the range will ever be used or referenced. Neither does such specification imply that any type of actual page is initially or ever associated with any virtual page in the range.

3.4.3 PAGE SHARING

If the task is to access shared-page regions of any type, the number of such regions the task will be sharing concurrently must be specified, when the task is cataloged. The number specified permits reservation of a Page-Sharing Extension of the task's TCB of the correct size. If the task has no such extension, no page-sharing operations are permitted.

If any page-sharing service is invoked to create or access a shared-page region and if a Page-Sharing Extension entry cannot be added, the requested service will not be performed.

3.4.4 MAX III-COMPATIBLE ALLOCATION

If the programs of a task intend to use the MAX III-compatible REX,ALLOCATE service, indicate, when the task is cataloged, the number of areas that will be simultaneously allocated through that service. The number permits the Compatible Allocation Extension of the task's TCB to be reserved and constructed.

If this service is invoked and the extension does not exist, the error exit address supplied is used as a return address.

3.4.5 TASK PUSH STACK SIZE/EXTENSIBILITY

A TASK PUSH STACK in the TCB is used by explicitly or implicitly-invoked reentrant system services to preserve your task's registers and as temporary operand storage. As these services are invoked, the amount of space used in the data structure varies according to the needs of the specific services.

When a task has execution control, no locations in this data structure are in use.

Each time a task executes a REX service call, the mechanism that connects the task to the service automatically saves the contents of all 15 general registers in the push stack. The service may also use a portion of the push stack for its own operands or intermediate register saving. If the service itself calls other REX services, another 15 locations are used to save registers again. Consequently, the number of locations that must be reserved for such usage is dependent upon the specific services that are invoked and the dynamic conditions that affect their execution and usage of other services.

All REX service mechanisms always assume that the stack is large enough so that any saves necessary can be performed. If an attempt is made to save information and enough room does not remain in the stack to save all of it, a condition arises that is termed "stack overflow". Under normal operating conditions, such an event will cause the task to be aborted.

Estimating the maximum size stack required by a task is either a time-consuming task that requires thorough knowledge of all of the services and their operations or a pick of a size that will always be large enough, even if a portion of the actual memory used by the task will be unused much of the time. To provide alternatives to either extreme, a memory management service is supplied, as part of the MAX IV Operating System, that can be implicitly invoked when necessary.

When generating a resident task at system generation or when cataloging a nonresident task, the user may specify:

- Whether or not the stack is "extensible".
- The number of words to be reserved for such usage in the task's TCB.

If the stack is not extensible, the first overflow of the stack causes the task to be aborted. If the stack is extensible, the first overflow causes an additional page (256 words) of MAP Ø virtual memory (with associated actual memory) to be concatenated to the existing stack and used as the push stack. When the amount of memory needed for the task's stack no longer requires use of the extension page, the page is disassociated from the task. If the task overflows the stack while the extension page is being used, the task is aborted.

Using this memory management service optimizes the task's usage of actual memory to the benefit of all tasks concurrently resident in the system. However, there is some system overhead involved each time the extension page must be obtained or released.

Use the following guidelines to decide what amount of overhead the task (with an extensible stack) can live with to obtain optimal usage of memory:

- If a Push Stack size of "Ø" words is specified, the efficiency of memory usage is maximized and the overhead associated with getting and releasing the extension page is incurred each time a task invokes a REX service.
- If a Push Stack size of "28" words is specified, the task is able to explicitly invoke almost all REX services under normal operating conditions without ever incurring the system overhead. (An exception to this statement is the REX,LOVER service which, in normal operation, requires a stack size of 7Ø words.)
- Considering the above, if a Push Stack size is specified at greater than the guidelines offered, the probability of incurring the system overhead and the probability of maximum memory usage efficiency is decreased.
- If a task is resident because it cannot afford memory not being available, the stack should be permanently sized according to worst-case usage.

3.5 MEMORY MANAGEMENT SERVICES

The MAX IV Operating System provides a task with memory management services in two forms:

1. As REX services that may be explicitly invoked by an executing program.

2. As Module Loader services that are implicit invocations of the REX services in the process of bringing a task or overlay from secondary storage to virtual memory.

The following paragraphs describe the explicit forms of the REX memory management services and the manner in which module loading invokes these services in the process of loading.

3.5.1 REX SERVICES

The following paragraphs summarize the function and usage of each of the REX memory management services that may be configured in the MAX IV Operating System. A complete description of each service is contained in the MAX IV EXECUTIVE REX SERVICES, System Guide Manual listed in the Preface.

3.5.1.1 ALLOCATE -- REX,#32/Call Number #29

This service causes a region to be modified such that:

- Local pages are associated with each non-shared virtual page of the region.

- All local pages of the region have associated AR's that permit Read/Write usage.

The region to be modified must lie totally within the virtual memory referenced as defined by the associated map image.

Any local pages associated with the region at the time the service is invoked remain associated with the region and, as necessary, additional available pages are associated with the region.

If local pages, with associated AR's permitting unrestricted usage, are associated with all virtual pages of the region when this service is invoked, the effect of invoking the service is the same as executing a "NOP."

3.5.1.2 ALLOCATE (MAX III-COMPATIBLE) -- REX,#29

In the MAX III Operating System, the REX Allocate service provides a contiguous area of "n" 128-word sectors. The MAX IV version of this service reduces the number of 128-word sectors requested to a number of pages specified by the user and attempts to find a free region of the task's (single) map where a region consisting of the required number of pages may be defined.

If such a contiguous region can be found by inspection of the associated map image, the required number of available pages is associated with the region. All associated pages have associated Access Rights (AR's) that permit unrestricted usage.

The address of the first virtual page of the region is passed to the invoking task after the region description has been recorded in the Compatible Allocation Extension of the invoking task's TCB. If such extension does not exist, the request cannot be honored and the task-supplied error routine will be activated.

3.5.1.3 DEALLOCATE -- REX,#32/Call Number #2A

This service causes all local pages in a specified region to be disassociated from the virtual memory of the invoking task and to be made available for reassignment.

The virtual pages with which local pages were associated become inaccessible to the invoking task. Any shared pages associated with the region remain associated and accessible.

If the specified region has no shared pages associated with it, the region becomes a free region after execution of this service.

3.5.1.4 DEALLOCATE (MAX III COMPATIBLE) -- REX,#2A

This service causes a region previously defined by an invocation of the compatible REX ALLOCATE service to become a free region. The private pages associated with the region become available for reassignment.

3.5.1.5 REX MODIFY AR's -- REX,#32/Call Number #57

This service causes the AR's associated with a region to be modified in accordance with the specified request.

While the CLASSIC 78xy hardware defines a degree of access of "no access permitted", the MAX IV memory management services do not permit such AR's to be associated with any virtual page that has an associated actual page. Instead, the memory management services reserve this degree of access to define free regions in a map.

This service unconditionally modifies the AR's associated with any page of a region provided that:

- The existing AR's do not indicate that an actual page is associated with the virtual page.
- The virtual page does not have a shared actual page associated with it.

This service also modifies the AR's associated with shared actual pages in a specified virtual region provided that the invoking task is permitted the degree of access requested. The degree(s) of access to the region permitted to the task is established at the time the task initially establishes its usage of the shared region or creates the region.

3.5.1.6 CREATE PRIVATE SHARED REGION -- REX,#32/Call Number #58

This service causes a private shared region to be defined for the use of a task and other tasks referencing the region. The region defined must have been accessible to the task as a local region prior to designating it as a private shared region.

The request for service will not be honored if:

- The task's TCB has no Page-Sharing Extensions available.
- The virtual region of the virtual memory is not already allocated as local pages.
- The name specified for the region duplicates that of an existing private shared region in the system directory of such regions.

Once a private shared region has been created, it may be referenced and accessed by any other task that uses the name of the region and can meet the criteria established for the region.

The region remains defined in the system directory as long as at least one task is accessing the region. When the last task accessing the region passes through the EXIT service, the definition is removed from the directory and the associated pages are made available for reassignment.

3.5.1.7 INSERT GLOBAL SHARED REGION -- REX,#32/Call Number #59

This service causes a global shared region to be made accessible to the invoking task in a specified free region of a specified addressing space of the invoking task.

The request for service will not be honored if:

- The task's TCB has no Page-Sharing Extension.
- The region, for which a beginning virtual address is specified, is not a free region as defined by the map image associated with the virtual memory referenced.
- The definition of the shared region cannot be established in the task's virtual memory, because the virtual addressing range of the specified addressing space is too small.
- The task cannot meet the access criteria applicable to the area.

3.5.1.8 INSERT PRIVATE SHARED REGION -- REX,#32/Call Number #5A

This service causes a single private shared region to be made accessible to a task in a free region of the task's virtual memory.

This service request will not be honored if:

- o A task's TCB has no Page-Sharing Extension.
- o The virtual region, for which a beginning virtual address is specified, is not a free region.
- o The region is not defined in the system directory of such regions.
- o A task cannot meet the access criteria applicable to the region.

3.5.1.9 INSERT SHARED LOAD MODULE - REX 32/Call Number #5B

This service causes a single shared load module region to be made accessible to a task in a free region of the task's virtual memory.

The request for service will not be honored if:

- o A task's TCB has no Page-Sharing Extension.
- o The virtual region, for which a beginning virtual address is specified, is not a free region.

If a task requests insertion of such a region and the region is defined in the directory but does not yet exist in actual memory, the necessary available pages are assigned and the module is loaded. If a task requests insertion and the module is being loaded, the task will be forced to relinquish, pending completion of loading and final definition of the region. If a task requests insertion and the region exists in actual memory, the area is made accessible to the task.

This service is similar in operation to a load-on-call type of service. However, because the module contents are accessible to all requesting tasks:

- o If the module contains one or more subprograms, these subprograms must be either reentrantly coded or serially reusable where such serial use is regulated by the subprogram contained in the module.
- o If the module contains only operands that are to be used as other than input constants, the tasks sharing the region will have to impose their own control on use of the area to avoid reference conflicts.

If no system directory of shared load modules exists, the module is loaded into your task's private memory.

3.5.1.10 EXTRACT SHARED REGION(S) -- REX,#32/Call Number #5C

This service causes one or more shared page regions to be made inaccessible to a task and ends the task's usage of such regions. This service may extract:

- A single named region of a specific type.
- Any or all regions of a single type within a specified region of a task map.
- Any or all regions of any type within a specified region of a task map.

Following execution of this service, any wholly shared-page regions extracted become free regions. If a virtual region is specified from which shared regions are to be extracted, any shared regions that lie either completely within or overlap the region are extracted. Local pages are not affected.

To ensure that a region is free, this service may be invoked requesting that all shared regions of any type be extracted for a specific region and immediately follow with an invocation of the REX,DEALLOCATE service to remove any local pages from the same virtual region.

3.5.2 SERVICES PERFORMED BY THE MODULE LOADER

The MAX IV Module Loader (ML) is capable of invoking the REX services in the process of loading modules. For a complete description of the Module Loader, refer to Quick Load Modules, chapter 8 in this manual.

Any module that is loaded by the loader always ensures that the virtual region it occupies is defined by invoking the REX ALLOCATE service. Because of this, it is never necessary to explicitly allocate the number of actual pages required by a task or its simple overlays since the loading process itself defines the actual page requirement for the module contents. The virtual region used by the module contents is always biased by the Load Offset, or "bias", specified for the loading process.

In creating the load module, memory management REX services may be invoked for virtual regions exterior to the module's loading region to:

- Cause shared regions to be inserted or extracted or erased where such regions are biased or unbiased with respect to the Load Offset specified for loading.
- Allocate or deallocate regions where such regions are biased or unbiased with respect to the Load Offset specified for loading.

For a description of the manner in which memory management services are explicitly invoked during the creation of a load module, refer to the TASK/OVERLAY CATALOGER, Programmer's Reference Manual listed in the Preface.

As an optional replacement for the module loader, the Sequential Task Loader provides the user with a limited subset of the task activation capabilities of the current module loader, while adding the capability to activate tasks stored on sequential media.

The Sequential Task Loader is added for users who are running process control systems in harsh environments, with the intention of permitting them to run MAX IV systems without discs. For example, a tape containing a MAX IV System and TOC GETs of all the tasks to run under the system is produced. The tape and a tape drive are taken to the CPU and the system booted. The X task invokes the Sequential Task Loader to load the tasks into memory. The tape drive is then disconnected and removed. This procedure minimizes the exposure of peripherals such as the tape drive to the harsh environment.

The standard module loader reads quick load modules which are set up for data-chain reads, which permits very rapid task activation. However, this makes it impossible to activate tasks off of sequential media. The essential difference between the module loader and the sequential task loader is that the latter does not do data chain reads, but rather normal sequential reads one record at a time. It decodes the LPRs in exactly the same way, but at the point where the current module loader does the data chain read, code to read the MORs sequentially is inserted.

The Sequential Task Loader is considerably less powerful than the module loader. In particular, it can only activate tasks that are present on the sequential device specified on the MLSEQUENTIAL statement in the SYSGEN, it cannot load overlays, and it will be much slower. These limitations are not significant in any system that would be running the sequential task loader because it is used only at system startup time to load the entire system.

3.6 SYSTEM CONSIDERATIONS IN MEMORY MANAGEMENT

In defining a task's memory management characteristics and its usage of memory management services, the operational characteristics of both the task and the other tasks concurrently resident in the system are affected.

The following paragraphs discuss the significant factors affecting the performance of both a task and the entire system.

3.6.1 NUMBER OF MAPS USED BY THE TASK

If the virtual memory requirements of a task offer a choice, consider the following when defining the number of addressing spaces or maps for the task:

- Every executing task uses 1 or 2 maps for execution.
- Maps never need to be unloaded since the map image reflects the contents of the map, so the contents can be used to reload the map. However, whenever a map is allocated to a task about to receive execution control (either because the task was never allocated a map or because an allocated map was taken for use by another task), the map must be (re)loaded.
- Given that the MAX IV Operating System permanently reserves MAP Ø for its own use, there exists only the following maximum sets of maps available for allocation to all tasks that are in the system and competing for execution control at any given moment in the CLASSIC 78xy computer:
  - Three large maps (1, 2, and 3) ... OR
  - Two large maps (1 and 2) and four small maps (4, 5, 6, and 7).
- If a task uses more than one map and there are a number of other tasks competing for execution control, the probability increases that:
  - One task forces another task to give up a map which slows down the other task's response time because its map has to be allocated and reloaded.
  - A task has its maps taken by another task, which slows down its response time.

3.6.2 MAP VIRTUAL ADDRESSING RANGE

The virtual addressing range used by a task in a map affects:

- The amount of TCB space required by a task (to hold the associated map image).

- The amount of time it takes to load the associated allocated map when the task is to receive execution control.

- The size of map -- large or small -- that must be allocated to describe the virtual memory during execution.

- The total number of actual pages that may be concurrently addressed by the task without modification of the map.

Again, it should be noted that the size of the virtual range specified for a map in no way implies usage of virtual pages in the range. It is certainly possible to have a task map that uses the virtual range from Ø to 65,536 and that never has actual pages associated with any virtual pages other than Virtual Page Ø and Virtual Page 255.

The modules loaded and the memory management services used, define the organization and usage of virtual memory. However, if execution and operation of a task is to proceed without System Protect traps occurring, the virtual range specified must be sufficient for all of the virtual memory required for the task.

3.6.3 LOCAL PAGE USAGE

After the MAX IV Operating System start-up or restart has occurred, the number of actual pages available for assignment as local pages is equal to the number of actual pages configured (currently accessible) in the CLASSIC system less those required for:

1. The MAX IV nucleus.
2. Hardware-reserved pages for dedicated purposes (Actual Pages Ø and 1).
3. Residence of tasks that are installation-specified to be resident after system start-up or restart.
4. Pages used for the Prologue TCB's of any permanently scheduled task
5. Actual pages reserved for the definition of global shared regions.
6. Pages used to contain optional MAX IV Operating System programs and their associated data areas.

The remaining actual pages form the pool of pages available for assignment to tasks. A task competes with every other task that is concurrently resident in the system for the assignment of available pages. Assignment of actual pages to all tasks is performed by the REX,ALLOCATE service that operates so as to assign needed actual pages to the highest (current) priority task and to force all other tasks requesting and needing actual pages to relinquish until the highest priority task's needs are satisfied.

Because the number of actual pages available for assignment is always limited, it benefits both the task and the system as a whole if the task keeps its usage of local pages and private shared regions at a minimum consistent with the requirements of the task structure and composition at any given moment.

3.6.4 SHARED REGION USAGE

Of the three types of shared regions, a task has no control over the definition and composition of global shared regions. These regions are defined and permanently reserved during the system generation process and are a constantly defined set of actual pages that are always unavailable for assignment to any task as local pages (although, of course, they can be associated with and used by any task permitted the degree of access requested).

However, the use of the other two types of shared regions permits a related group of tasks to minimize their over-all total local page requirement.

As a control on the ability of any system of tasks to completely use an unlimited number of actual pages for use as shared regions, the system generation process permits a limit on the number of actual pages to be used for such purposes concurrently to be specified.

The system generation process also permits the installation to specify how many of each type of region may be currently defined by specifying the amount of space (in MAP Ø) to be reserved for definitions of regions of each type.

When a task (or its loading) calls for creation of a private shared region or insertion of a shared load module, the limits specified by system generation are inspected by the applicable REX services to determine if the request can be honored at the current instant (or ever).

3.6.5 TASK ROLLABILITY

When creating a task, there is one option that does not affect the task's memory management operations but does affect the memory management operations of the MAX IV Operating System. A task may be "rollable" or "non-rollable."

There exists an optional extension of the REX,ALLOCATE service called the "Roller." The Roller is invoked -- when configured -- when a task requesting and needing actual pages cannot obtain the required number of pages from the pool of pages available for assignment. The REX,ALLOCATE.service then attempts to find a task in the system that has all of the following qualities:

- Is active
- Has the lowest priority less than that of the task requesting the pages
- Has local pages assigned
- Is rollable

Upon locating such a task, the REX,ALLOCATE service then invokes the Roller to "roll out" the task. The contents of the actual pages are moved to secondary storage upon completion of any I/O operation associated with the selected task. Following the movement of the task to secondary storage, the local pages of the selected task are placed in the available page pool.

The process of task selection and roll-out continues until either enough pages are made available to satisfy the requirements of the requesting task or no more tasks can be selected for roll-out.

The only effect of the rolling operation on any task is that the time between system entry and exit is lengthened (the task's system resource usage characteristics do not change).

The effect of having the Roller configured, then, is to increase the number of pages available for higher priority tasks at the expense of lower priority tasks. This improves system operating efficiency, so it is to the installation's benefit to have all but the highest priority task(s) created as rollable.

3.7 THE ROLLER

The ROLLER is an optional MAX IV task. It is present in the system only if it is specified during system generation.

The primary function of the ROLLER is to make actual pages of private memory, which have been allocated to one task's instruction space and operand space, available to another task. The ROLLER performs this function by moving the contents of the task's instruction space and operand space to disc and returning the actual pages to the available memory pool. If the Shared Page Bit is set, the roller does not roll that page.

The ROLLER performs automatically whenever a request for memory is made. The ROLLER rolls out tasks that are currently rollable and are also of a lower priority of the task requesting memory in the CPU queue until the requested memory is obtained.

The presence of the ROLLER is specified during system generation by adding the following specification statement to the SYSGEN Task Block.

```
                1          2          3
ROLLER [priority], [MAP[PED]], file  REQUEST ROLLER TASK PRESENCE
```

Parameter 1 is the optional priority of the ROLLER task. If no priority is specified then the ROLLER task will dynamically change its priority to match that of the task that is to roll in or out. If a priority is specified then the ROLLER task remains fixed at that priority level. Parameter 1 is ignored if the Non-MAP Ø Resident Roller is optioned. Instead of using this parameter, the Non-MAP Ø Roller can be cataloged with the task variable PRI to have it remain fixed in the CPUQ.

Parameter 2 is optional. If MAP[PED] is specified, the Non-MAP Ø Resident Roller is configured. For the MAP Ø Resident Roller, this parameter must be defaulted.

Parameter 3 specifies the logical file name from which the Non-MAP Ø Resident Roller is to be activated. Default is SM. This parameter is ignored if the MAP Ø Resident Roller is optioned.

The association between the logical global Roll file RO and a disc-device QRO is specified during system generation by adding the following specification statement to the SYSGEN Task Block as part of the System Exceptional Condition Task definition:

```
LOGFILE        RO,QRO
```

The disc-device QRO must be a fixed, single-extent partition (contiguous) with a geometry of 1 and a record size limit of Ø.

The following statement added to the Basic I/O System SYSGEN Block after the transport definition statement for M1 defines a Roll File partition of 4ØØ tracks on M1.

```
DISCDEVICE     QRO,M1,4ØØ,1,Ø,Ø
```

The required number of tracks necessary for the Roll file may be calculated as follows:

$$\#\ TRACKS = \frac{WRDS}{S * W} + (3 * NTKS)$$

where:

WRDS = maximum number of words ever to be rolled out at any point in time.
NTKS = maximum number of tasks ever to be rolled out at any point in time.
S = number of sectors/track on the transport.
W = number of words/sector on the transport.

3.7.1 MAP Ø RESIDENT ROLLER

On system start-up the prescheduled resident Roller task "ROL" loads and executes the Roller Initialization Overlay (ROLIØ1 on the SM file) to initialize the Roller data structures. The Roller data structures are built in a MAP Ø system page and are pointed to by the Roller Extension Block word "UCBRLE" in the Upper Control Block. During initialization, actual page(s) are acquired (and recorded in the RLE) in order to build and maintain a linked list of available Roll File Granules of the global file RO. Roll file granularity is one track and the link list consists of one non-zero word (bits Ø - 1, bits 6-15 point to the next available granule) for each available granule. The Roll file RO may consist of from 1 to 1Ø24 granules. The Roller Initialization Overlay loads the Roller Task Overlay ROLTØ1 from the SM file and transfers control to it. The Roller Task Overlay contains the Roller Algorithm Code.

The Roller task code that performs the Rolling and the task Roll file return code, is needed to return the roll file tracks to the available list when a rolled task is killed by another task.

3.7.2 NON-MAP Ø RESIDENT ROLLER

During system start-up the MAP Ø Page for the Roller's Upper Control Block extension is acquired. The prescheduled Roller Task "MP.ROL" begins execution and initializes the data structures, as the MAP Ø Roller does, but does not load any overlays (all code is contained within the task's body). An additional word (Word Ø) within the Roller's extension is used as a status word. Bit Ø of this word is set when the Roller task has completed initialization.

I/O operations that are queued to a task that is to be rolled are allowed to complete while the task is rolled out because the task's I/O buffers and UFT's are not rolled if they are associated with a queued I/O operation.

Any task that has I/O done into or out of its Instruction or Operand space by another task must be cataloged as unrollable.

Neither the Roller Task "ROL" nor the "MP.ROL" task may be killed.

Rolled tasks may be killed.

Rolled tasks may have their priorities and positions changed in the CPUQ.

The assignment of the global file RO may not be changed once the ROLLER task is executing.

Tasks that are cataloged as unrollable are rollable until they are completely loaded.

The Roller is intended to operate in an environment where the Roll file will not become full. If the Roll file becomes full a "Roll File Full" message is output. If this occurs the Roll file should be enlarged in order to avoid system deadlocks.

Both the MAP Ø Resident and Non-MAP Ø Resident Roller tasks have a variables extension. If during initialization a task variable named PRI is present then the ROLLER task sets its MAP task system option bit and changes the ROLLER task priority to the value of the task variable PRI. This fixes the position of the ROLLER task in the CPU queue. If the task variable is not present then the ROLLER task dynamically changes its priority to match that of the task that is to roll in or out.

3.7.3 ROLLER MEMORY ALLOCATION ALGORITHM

The Roller Memory Allocation Algorithm consists of three parts:

1. Task Memory Allocation
2. Task Memory Deallocation
3. Roller Roll in

3.7.3.1 Task Memory Allocation

If the Roller is present, the allocation part of the algorithm is entered every time a request for one or more pages of memory is made.

If enough memory is available in the currently available pool, a return from the algorithm is made to the REX,ALLOCATE service where the memory allocation occurs.

If enough memory is not currently available and the caller is impatient, a return is made to the REX,ALLOCATE service with no further action by the Roller. The REX,ALLOCATE service then sets the condition codes and returns to the caller.

If enough memory is not currently available and the caller is not impatient, an attempt is made to acquire memory by rolling out tasks that are below the caller in the CPUQ. If the request is for only one page - possibly an emergency situation - regardless of whether or not a higher priority task is awaiting memory allocation, the one page is allocated from an emergency supply of pages if it is not yet exhausted. If the request is for more than one page and a higher priority task is already awaiting memory allocation, the requesting task relinquishes until it becomes the highest priority task awaiting memory allocation at which time it acquires its memory or forces tasks that are lower in the CPUQ to be rolled out until enough memory becomes available.

3.7.3.2 Task Memory Deallocation

If the Roller is present, the deallocation part of the algorithm is entered every time one or more pages of actual memory have been deallocated.

If the emergency page supply is low, pages are moved from the currently available pool to the emergency supply until either the emergency supply is replenished or the currently available pool is exhausted.

3.7.3.3 Roller Roll In

The Roller Roll In part of the algorithm is entered as part of the Roller task anytime the Roller finds that it is time to service another task's requirement to roll in. If enough memory is not currently available to permit the rolled out task to be completely rolled in, an attempt is made to find a task lower in the CPUQ to roll out in order to make enough memory available for the roll in.

3.7.4 ROLLER DYNAMIC RECONFIGURATION

The ROLLER task has the capability of detecting a ROLLER task exit request from the system operator. The request is posted by setting the value of the ROL task's unspecified task variable name to equal the CAN code value "EXI". This is easily accomplished by the following operator communications sequence:

```
/ROL/REL EXI              MAP Ø Resident Roller
or
/MP.ROL/REL EXI           Non-MAP Ø Resident Roller
```

Upon sensing this value in the unspecified task variable name the following occurs:

1. When the number of roll file granules currently available equals the number of roll file granules initially available (the roll file is empty) then the ROLLER task prepares to exit.

2. After this point any roll requests posted will not be acted upon until the ROLLER task is again activated.

3. The MAP Ø Resident Roller task unlinks and deallocates its UCBRLE extension.

4. The Non-MAP Ø Resident Roller resets Bit Ø in Word Ø of UCBRLE extension and returns any pages in the emergency page pool and the actual pages used for the linked list of roll file granules.

5. The roller task will then output the following messages to the global CO file:

```
ROLLER LEAVING SYSTEM BY OPERATOR REQUEST
ROLLER EXITING
```

Following this the global file "RO" (the roll file) may be reassigned to a larger or smaller disc partition if needed. At some point afterward the system operator may start the ROLLER task running by the following operator communications sequence:

```
/ROL/ACT  MAP Ø roller
or
/MP.ROL./ACT   Non-MAP Ø Roller
```

The ROLLER task then performs its normal startup. The ROLLER task need not be re-activated by the system operator unless it is needed in the system.

3.7.5 ROLLER INFORMATIONAL MESSAGES

All messages output by the ROLLER task are sent to the global CO file.

The following message is output after the ROLLER task has unlinked and deallocated its data structures but prior to exiting the system.

```
ROLLER EXITING
```

3.7.6 ROLLER ERROR MESSAGES

The following message is output when the MAP Ø Resident Roller cannot successfully allocate its dummy map image page:

```
ROL ERROR:  MM$OSR RETURNED #hhhh
```

("#hhhh" is the error code returned by MM$OSR)

The following message is output if the MAP Ø Resident Roller cannot successfully load one of its overlays:

```
ROL ERROR:  LOAD OF OVERLAY xxxxxx RETURNED COND CODES #hh
```

("xxxxxx" is the name if the overlay and "hh" is the condition codes value returned by the load overlay REX).

The following message is output when the logical file RO is either unassigned or not assigned to a disc device:

```
RO FILE UNASSIGNED OR NOT ASSIGNED TO DISC
```

The following message is output when the disc device to which the logical file RO is assigned has a geometry other than 1 or a record size limit other than Ø.

```
RO FILE GEOMETRY NOT ONE OR RECORD SIZE LIMIT NOT ZERO
```

The following message is output if the REX allocate call to allocate pages into the dummy map image returns in error:

```
DUMMY MIAP ALLOCATE ERROR
```

The following message is output if a data integrity error occurs while reading data from the roll file:

```
RO FILE FPI NOT = DPI IN RECORD
```

The output of this message puts the ROLLER task in OC hold. If the ROLLER task is resumed at this point it will repeat the read and verify operation.

The following message is output by the Non-MAP Ø Resident Roller if the MAP Ø page used for the Roller UCB extension was not acquired during startup:

NO ROLLER USED CONTROL BLOCK EXTENSION

The following message is output by the Non-MAP Ø Resident Roller if it is activated and the MAP Ø Resident Roller is present in the system:

MAP Ø RESIDENT ROLLER CONFIGURED

3.8 MEMORY INITIALIZATION

The bootstrap loader (SAL) and coldstart initialization ensure correct parity is written to all of functional memory.

The bootstrap loader (SAL) initializes all of the first 64K memory locations that read with bad parity by re-writing them with correct parity.

MAX IV coldstart initialization performs a similar function for the rest of the functional actual memory locations.

3.9 EXTENDED MEMORY ON A CLASSIC 78xy COMPUTER

Extended memory is handled as an extension to the Global Shared Regions processing. An extended memory region must be a global shared region and must be specified as such during System generation. An extended memory region consists of contiguous regions of 8K blocks of actual memory and must begin on an 8K boundary. A task may "insert" only one extended memory region at a time. Inserting a second extended memory region forces any previously inserted region to be extracted. An extended memory region is not actually inserted into the tasks normal 64K word operand space but instead is a non-contiguous extension of the task's operand space. Access to the region is made by executing an Extended Memory Access instruction (EXMA) before the memory read or write instruction.

3.10 CLASSIC 78xy COMPUTER SHARED MULTIPORT MEMORY

The Shared Multiport Memory (SMM) is a CLASSIC hardware option that allows multiple CPU access to a common memory subsystem. Because the SMM access time is somewhat slower than the CLASSIC's local memory, the Operating System limits its use to operand space, and requires that all SMM be allocated at system generation as either Global Common or Extended Global Common. User access to operands in the SMM may be by virtual addressing or extended memory addressing modes.

CHAPTER 4
INTERRUPT AND TRAP PROCESSING

This chapter explains the handling of hardware traps and interrupts by the MAX IV Operating System. It contains three sections:

- Interrupt Level Ø - Power Failure and Recovery Interrupt
- Interrupt Level 2 - Software Protection Trap
- Interrupt Level 4 - The Unimplemented Instruction Trap

This chapter explains the function and purpose of the above interrupt or trap handlers; the information contained in the registers, stacks, and PS/PR; the condition codes set; and the user hooks provided.

4.1 INTERRUPT LEVEL Ø - POWER FAILURE AND RECOVERY INTERRUPTS

The MAX IV power failure and recovery interrupt routine (IØ.PWR) processes changes in the power state of a CLASSIC CPU, and a Shared Multiport Memory (SMM) rack. All registers are saved and restored. Power events are processed based on the current power state existing prior to the interrupt and on the power event that just occurred. A user hook is provided for each possible power event. Refer to the chapter on user hooks in the MAX IV DATA STRUCTURES System Design Manual.

The CLASSIC CPU has up to four possible power states. The number of power states supported is dependent on both the model number of the CLASSIC and the type of power supply with which it is equipped. It can distinguish between AC current and DC current state changes. Changes in AC current are early warning or brown out conditions. The CPU continues to run. Changes in DC current are absolute power failures and recoveries. The CPU halts and restarts. When a power failure occurs, the level Ø interrupt request latch takes up to 5Ø microseconds to clear. IØ.PWR must loop waiting for the request to clear prior to processing the interrupt. The type of power event is returned in the condition codes. The power event is compared to the current power state (LCBMST) to determine the type of processing required. An unexpected interrupt, such as receiving a power up interrupt when power is already up, is discarded. Normal processing is resumed.

A CLASSIC CPU can support two remote SMM racks. These have a separate power supply. Changes in power state are reported separately from the CPU power supply. Two states are reported, remote DC up and down. Remote AC power states are not reported.

In order to conserve MAP Ø space, only local DC power interrupts are handled. To expand the Level Ø Handler to cover AC lowline and recovery and remote DC failure and recovery, specify the following in the SYSGEN: SYSOPT,,,PWR

4.1.1 POWER EVENT

A power event is the cause of a level Ø interrupt. The type of power event is returned in the condition codes for a CLASSIC CPU only. Conditions codes are:

CCN 1= LOCAL POWER EVENT (CPU)
Ø= REMOTE POWER EVENT (SMM)#

CCZ 1= POWER FAILURE
Ø= POWER RECOVERY

CCO 1= AC CURRENT#
Ø= DC CURRENT

POWER EVENT	POWER DESCRIPTION	CONDITIONS CODES (NZOC)	USER ROUTINE
LOCAL DC DOWN	TOTAL POWER FAILURE CPU DC FAILURE	'1100	U$POWF
LOCAL DC UP*	CPU POWER RECOVERY CPU DC RECOVERY	'1000	U$POWR
LOCAL AC DOWN#	EARLY WARNING FAILURE BROWN OUT AC LOWLINE CPU AC FAILURE	'1110	U$EWPF
LOCAL AC UP#	EARLY WARNING RECOVERY CPU AC RECOVERY	'1010	U$EWPR
REMOTE DC DOWN#	SMM POWER FAILURE	'0100	U$RPWF
REMOTE DC UP#	SMM POWER RECOVERY	'0000	U$RPWR

Table 4-1. Power Events

*When a CPU loses its DC power, the system is considered totally down regardless of a remote power supply state. When CPU power is restored, only the CPU is considered up. The state of a remote power supply is unknown and assumed down. A separate REMOTE DC UP power event is expected.

If a SYSGEN does not contain a SYSOPT,,,PWR macro, AC power interrupts are ignored. User hooks U$EWPF and U$EWPR do not exist.

4.1.2 CURRENT POWER STATE

The current power state is the condition of the power supplies prior to receiving a power event. A computer has a power supply at the CPU. If an SMM is connected to it, the SMM will have its own power supply. CPU and SMM power events are reported separately and are referred to respectively as LOCAL and REMOTE power events. The current power state is maintained in LCBMST and is updated each time a power event occurs.

4.1.3 USER ROUTINES

A user routine exists for each of the six power events. All registers are saved prior to calling the user routine. The current power state is in LCBMST. The power event is in Register 4. The new power state upon completion of the interrupt handling is in Register 3. Register 1 is the link back to IO.PWR.

For examples of basic user routines refer to the U$COMM User Hook section of the Terminal Monitor Program Chapter of this manual.

4.1.4 POWER EVENT PROCESSING

The following table lists the various power states and the appropriate processing.

POWER EVENT	PROCESSING
LOCAL DC DOWN	This is considered total power failure. The current power status in LCBMST is set to total power failure. The power failure bit in LCBSTA is set. The power failure bit in the X task is set. U$POWF is processed. The system is halted.
LOCAL DC UP	When power is restored to the CPU, the current power status is set to CPU power up, remote power down. U$POWR is processed. The level Ø interrupt vector (#2Ø) is set to the system restart address (#1). The system is restarted.
LOCAL AC DOWN#	When the AC current of the CPU drops below its lowline threshold, the current power state is set to early warning failure. The AC lowline occurred bit is set in LCBSTA. U$EWPF is processed. Control returns to the point of the interrupt.
LOCAL AC UP#	When the AC current of the CPU is restored, the current power state is set to early warning recovery. U$EWPR is processed. Control returns to the point of the interrupt.
REMOTE DC DOWN#	When the power fails, all global shared regions defined in that rack are set offline. All tasks using those failed global shared regions are aborted with the following message: !ABORT (PWR SMM) or U$RPWF is processed. The current power state is set to remote power failure. Control is returned to the point of interrupt.
REMOTE DC UP#	When an SMM rack has its power restored, all global shared regions defined in that memory are set online. U$RPWR is processed. The current power state is set to remote power recovery. Control returns to the point of interrupt.

Table 4-2. Power Event Processing

\# If the SYSGEN does not contain a SYSOPT,,,PWR macro, AC power interrupt support is removed.

4.1.5 COLD START AND RESTART INTIALIZATION

Upon entry into the cold start initialization routine (CS.INI), the system restart address (#1) is defined to be CS.INI. If a total power failure occurs before CS.INI completes, the system restarts at CS.INI when power is restored. Once cold start completes, the restart vector is set to the restart initialization routine (RS.INI).

The Plane Status Table (PLE) and the Global Shared Regions Table (GSR) are initialized by CS.INI. If the SMM is not powered up at the time of cold start, table entries are not built nor recognized by the system.

4.2 INTERRUPT LEVEL 2 - SOFTWARE PROTECTION TRAP

The MAX IV software protection trap (I2.STR) processes access rights violations, privileged instruction violations, and CLASSIC instruction stalls. It uses Register Block #E to save and restore the values in Register 1-15 upon entry into the trap. Location #24 contains the address of the violating instruction. Location #45 contains the program status word at the moment of the violation. The condition codes at the time of the trap are in location #45.

4.2.1 STALL ALARM

If an instruction on a CLASSIC CPU stalls, the software protect trap routine is entered by a level 2 hardware interrupt. The trap routine increments the stall count (I2$STL) and the instruction is retried.

4.2.2 SYSTEM HALTS

The software protection trap causes the system to halt if one of four cases exists:

- o An unconditional halt
- o A trap with another interrupt active
- o A system-caused violation
- o If no task was executing at the time of the trap

In all of these cases the PR contains the address of the halt instruction (I2$PTH) within the protection trap routine (I2.STR). The PS contains an Instruction Map of Ø, an Operand Map of Ø, the current register block, the privileged bit set, and conditions codes signifying which of the four cases caused the halt.

If the HALT/RUN switch is toggled, the protection trap routine treats the protection violation as a user task protection violation where no user trap is provided.

4.2.3 UNCONDITIONAL HALT

By setting Bit 15 of (CCC) of location #44 the system always halts if a software protection violation occurs. The PR and PS contain values explained in the previous section called SYSTEM HALTS. The condition codes are set to 15 (NZOC = '1111).

4.2.4 TRAP WITH ANOTHER INTERRUPT ACTIVE

Upon entry into the protection trap routine, if any other hardware interrupt level is active (other than level 2), the system halts. The PR and PS contain values explained in SYSTEM HALTS above. The condition codes are set to 7 (NZOC = '0111).

4.2.5 SYSTEM PROTECTION HALT

If the MAX IV Operating System causes a protection violation, the system halts with a PR and a PS explained the previous section called SYSTEM HALTS. Condition codes are set to 3 (NZOC = '0011).

4.2.6 NO TASK RUNNING HALT

If no task is currently executing in the CPU queue because the TASKMASTER is active (if bit 15 (CCC) of the TASKMASTER status word (HCBTMS) is reset) when the protection violation occurred, then the system halts with a PR and a PS as explained in the previous section called SYSTEM HALTS. The condition codes are set to 1 (NZOC = '0001).

4.2.7 TASK PROTECTION VIOLATIONS

The software protection trap routine processes violations caused by tasks without halting the system. The standard recovery is simply to abort the violating task and print a message. The user may, however, provide a user protection trap routine to process the violation in some other way. (Refer to the USER TRAP REX Service in the MAX IV EXECUTIVE REX SERVICES, System Guide Manual listed in the Preface.)

4.2.8 STANDARD TASK PROTECTION TRAP RECOVERY

If no user trap routine exists, the task is aborted. An abort message is printed along with the address of the violation.

EXAMPLE

```
!ABORT (PRO 1234*) /B  /ABC
```

The task's interrupt PR and PS words (TCBIPR and TCBIPS) are set to the violating PR and PS. If the task's PS/PR stack is empty, these values are the same as the values in #24 and #45. If the task's PS/PR stack is not empty as would be the case when in REX service, the interrupt PS and PR are set to the values of the first entry on the PS/PR stack. Registers 1-15 are restored to their values at the time of the violation.

4.2.9 USER PROTECTION TRAP RECOVERY

If the user has provided a protection trap routine, I2.STR returns to that routine. If the task's PS/PR stack is empty, location #24 is set to the address of the user's routine, Registers 1-15 are restored to their values at the time of the violation, the Level 2 interrupt is cleared and control is passed to the user's routine.

If the PS/PR stack is not empty, the stack is emptied. The condition codes upon entry into I2.STR are combined with the resources in TCBRES to form the PS returned to the user trap routine. The task's main stack is cleared except for the first 15 register values. These register values are returned to the user trap.

4.3 LEVEL 4 - UNIMPLEMENTED INSTRUCTION TRAP

Interrupt Level 4 (the Unimplemented Instruction Trap) causes a transfer from the task's virtual address space into the system's address space (MAP Ø) and back to the virtual address space of the task. The service of the instruction that caused the trap is not performed prior to the interrupt. MAX IV uses this type of trap processing to execute REX services that are requested by a task but executed at the system level. For more information refer to the MAX IV EXECUTIVE (REX) SERVICES System Guide Manual listed in the Preface.

CHAPTER 5
TERMINAL MONITOR PROGRAM (TMP)

The MAX IV Terminal Monitor Program (TMP) provides a set of features used to control terminals and programs. A user can access TMP by the BREAK key or CONTROL A (called an "attention request"). TMP provides a set of directives that control the execution of tasks. An additional feature of TMP is the control of Ring Detect supported communications lines. Refer to the Asynchronous Communications Handler chapter in the MAX IV Communications Handlers System Guide for attention requests of particular controllers.

TMP detects attention requests, processes commands, and sends responses for several terminals simultaneously. When an attention request is made, TMP takes exclusive use of the terminal. This assures the user of being able to interrupt a program in order to control its execution. TMP does not process commands itself, but calls in overlays to do the work.

NOTE: No task should attempt to take exclusive use of a terminal used by TMP.

An online programmer or user can use TMP to start, stop, make file assignments for, and otherwise control the execution of a task, without affecting other tasks in the system. System operators and system programmers may use TMP to gain access to the "richer" set of Operator Communication directives from a remote terminal (refer to the MAX IV OPERATOR COMMUNICATIONS System Guide Manual listed in the Preface).

TMP runs as a privileged Non-MAP Ø task; it must not be specified as RESIDENT during SYSGEN. Special requirements for cataloging TMP are described in the next section.

5.1 CUSTOMIZING TMP

TMP is designed to run under any sort of configuration. In order to perform effectively, TMP requires some information describing the environment.

5.1.1 SYSGEN

Each terminal that will have access to TMP requires a LINEMONITOR statement in the SYSGEN. (Refer to the MAX IV SYSGEN System Guide Manual listed in the Preface.)

TMP has the capability of searching a user SM file before searching the standard SM file. To use this capability, a load module file should be created. The assignment of the global logical file for USM should be modified in the SYSGEN to assign USM to the user load module file. The USM file will be searched before the standard SM file is searched to find the requested overlay.

5.1.2 TASK VARIABLES

TMP refers to five task variables for operational information. If the default values are not satisfactory, then TMP should be recataloged with new parameters.

1. BUF - Number of I/O Buffers

   The number of terminals with which TMP can simultaneously interact is determined by the number of buffers allocated to it. TMP uses the value of the task variable 'BUF' as the number of buffers to allocate. This value may be specified when TMP is cataloged with the following statement:

   ```
   VAR   BUF   n     ('n' is the number of buffers.)
   ```

   If this statement is missing, a default value of 2 is used.

   In order to input a command, TMP must assign a buffer to a terminal as soon as an attention request is recognized. Since a terminal operator might not in all cases transmit a command right away, it is recommended that several buffer areas be specified so that one user does not prevent other users from accessing TMP.

2. OVL - Number of Commands to Process Simultaneously

   The number of commands that TMP can simultaneously process is determined by the number of overlay processing areas (OPA) allocated to it. TMP uses the value of the task variable 'OVL' as the number of OPA's to allocate. This value may be specified when TMP is cataloged with the following statement:

   ```
   VAR   OVL   m     ('m' is the number of OPA's.)
   ```

   If this statement is missing, a default value of 1 is used.

   Normally, one area guarantees a reasonable response time. However, if commands that generate several lines of output, particularly to slow terminals, are heavily used, then more than one area should be specified.

3. FIL - File Name for Command-processing Overlays

   TMP loads in an overlay to process each command. The file used by TMP to find overlays may be specified at catalog time with the following statement:

   ```
   VAR   FIL   nam   ('nam' is the CAN code file
                      name.)
   ```

If this statement is missing, the default is the SM file. USM will be searched, if assigned, prior to searching SM to find the requested overlay.

4. WPT - Write Prompt Timer Flag/Value

Whether or not the I/O operation that writes the prompt character (">") to the terminal is internally timed is specified through this task variable. TMP uses the value of the task variable 'WPT' as the number of ticks within which the I/O operation should complete. This value may be specified when TMP is cataloged with the following statement:

```
VAR   WPT   t  ('t' is the number of ticks.)
```

If this statement is missing, the I/O operation is not timed (timer missing, timer disabled). If the statement is present, it has the following meaning:

< Ø or = Ø	Do not time the operation of writing a prompt (timer present, timer disabled).
> Ø	Time the I/O operation of writing a prompt using 't' ticks as the timer value (timer present, timer enabled).

Note that the task variable 'WPT' may be modified while TMP is executing to enable or disable the timer. The variable is checked every time a break is processed. The timer is useful for causing TMP to give up attempting to write a prompt when circumstances arise that cause the I/O operation not to complete. An example of such a circumstance is a terminal that generates a BREAK when the terminal is powered off. The I/O operation remains busy until the terminal is powered on. This would normally use up an I/O node, a buffer control block and a buffer area that would be unavailable to other users. If the operation is timed, some subsequent activity by TMP (either another I/O operation completing or a new user issuing a BREAK) would cause TMP to make these resources available (if the specified timer expired). A value of t = 3ØØ ticks (3 seconds on a system with 1ØØ ticks per second) is recommended.

5. RCT - Read Command Timer Flag/Value

Whether or not the I/O operation that reads a command from the terminal (after the prompt character is sent to the terminal) is internally timed is specified through this task variable. TMP uses the value of the task variable 'RCT' as the number of ticks within which the I/O operation should complete. This value may be specified when TMP is cataloged with the following statement:

```
VAR   RCT   t  ('t' is the number of ticks.)
```

If this statement is missing, the I/O operation is not timed (timer missing, timer disabled). If the statement is present, t has the following meaning.

< Ø or = Ø	Do not time the operation of reading a command (timer present, timer disabled).
> Ø	Time the I/O operation of reading a command using 't' ticks as the timer value (timer present, timer enabled).

Note that the task variable 'RCT' may be modified while TMP is executing to enable or disable the timer. The variable is interrogated every time a break is processed. The timer is useful for causing TMP to give up waiting for a command to be input after a prompt is sent to the terminal. Since the reading of a command means an I/O node, a buffer control area, a buffer, and (possibly) an overlay control area are in use, these resources would normally be unavailable to other users until the user ends communication with TMP by transmitting a blank line, or by entering a command that releases the terminal after processing. If the operation is timed, some subsequent activity by TMP (either another I/O operation completing or a new user issuing a BREAK) would cause TMP to make these resources available (if the specified timer expired).

5.1.3 I/O OPERATIONS

When cataloged, TMP must be given two vacant logical file entries for each terminal it may simultaneously process. This is done with a LOGFILE statement.

TMP must be cataloged with one I/O node for each terminal it may simultaneously process. This is done with an IOPERATIONS statement.

5.1.4 CATALOG OPTIONS AND CONSIDERATIONS

When cataloging TMP the only option allowed is the GO option. Setting any other option may cause an error.

The GO option causes the body of TMP to leave memory when TMP is not in use, only the TCB remains. If the GO option is set, then TMP should be cataloged as prescheduled and unabortable.

Since Terminal Control Lists (a MAX IV data structure required for break support) contain the TCB address of the break support task, it is highly recommended that the TMP task have a permanent prologue TCB address. This prevents a non-break support task from using the same TCB address if TMP exits. Permanent prologue TCB addresses are achieved by prescheduling TMP with the SYSGEN PRESCHEDULE statement and specifying PERMANENT for the appropriate parameter. (See the MAX IV SYSGEN System Guide Manual listed in the Preface.)

TMP requires additional unallocated virtual addressing space in order to dynamically allocate memory for overlay control blocks, buffer control blocks, buffers, and for loading the overlays. The SPACE Directive in the Task Overlay Cataloger (TOC) Processor should be specified for additional space as follows:

```
SPACE 0,xxxx
```

('xxxx' is the additional required number of words of unallocated virtual space computed as a function of the task variables @OVL and @BUF and the TMP equates of OCBLN, BCBLN, and BUFLN (see Appendix G for the values of these equates).) Specifically,

```
xxxx = (OVL*(OCBLN+512) + BUF*(BCBLN+BUFLN) + 255)
```

5.1.5 U$COMM USER HOOK

The U$COMM user hook is called by TMP (by routine BS$WHY) after determining that a line monitor event has occurred. This allows the user to do any preprocessing that may be necessary before TMP takes over to complete its normal functions.

The U$COMM routine supplied with MAX IV follows:

```
      PGM     U.COMM
      INS*    MC,IVSEQU
      INS*    MC,IVUEQU
      INT     U$COMM
*     THE U$COMM USER HOOK IS CALLED BY TMP (BY SUBROUTINE BS$WHY)
*     AFTER DETERMINING THAT A LINEMONITOR EVENT HAS OCCURRED.
*
*     THE FOLLOWING INFORMATION IS PASSED TO U$COMM:
*         R3 =  TERMINAL CONTROL LIST (TCL) ADDRESS
*         R6 =  RETURN LINK : BYPASS NORMAL FUNCTIONS
*         R7 =  RETURN LINK : NORMAL RETURN
*         R9 =  STATUS AND RETURN FUNCTIONS WHERE:
*               B0  = CURRENTLY PROCESSING A BREAK EVENT
*               B1  = CURRENTLY PROCESSING A RING EVENT
*               B2  = CURRENTLY PROCESSING A DISCONNECT EVENT
*               B3 - B4 = RESERVED
*               B5  = STATUS OF CLEAR TO SEND (CTS) MODEM SIGNAL
*               B6  = STATUS OF CARRIER DETECT (CD) MODEM SIGNAL
*               B7  = STATUS OF DATA READY (DSR) MODEM SIGNAL
*               B8 - B10 = RESERVED
*               B11 = ESTABLISH ATTENTIVE TASK FUNCTION (RING)
*               B12 = ACTIVATE ATTENTIVE TASK FUNCTION (RING)
*               B13 = DEESTABLISH ATTENTIVE TASK FUNCTION (DISC)
*               B14 = KILL ATTENTIVE TASK FUNCTION (DISC)
*               B15 = PREPARE FOR COMMAND INPUT FUNCTION (BREAK)
*         R10 = OUTPUT DEVICE NAME
*         R11 = INPUT DEVICE NAME
*         R12 = LOAD MODULE FILE NAME (DEFAULT = @BM)
*         R13 = TCL STATUS WORD AT EVENT TIME
*         R14 = ATTENTIVE TASK NAME (MSH)
*         R15 = ATTENTIVE TASK NAME (LSH)
*
```

```
*     *****CAUTION  ***** CAUTION ***** CAUTION ***** CAUTION*****
*     *****                                                *****
*     *****THE USER MUST NOT DESTROY REGISTERS 10,11, AND 13 *****
*     *****AND MUST NOT MODIFY BITS 0,1, AND 2 OF REGISTER 9 *****
*     *****
*     *****CAUTION *****  CAUTION ***** CAUTION ***** CAUTION*****
*
U$COMM EQU       $
       BRX,R7          RETURN - CONTINUE WITH NORMAL FUNCTIONS
       END
```

5.2 INVOCATION AND SHUTDOWN OF TMP

Communicating with TMP begins by initiating the appropriate attention request. When the request is recognized, TMP sends a prompt character (">") to the terminal and prepares to read a command.

Enter TMP commands in response to TMP's prompt, ">". The format and description of these commands are in the following pages.

Unlike the OC task, TMP does not release the terminal after a timeout period. The user may end communication with TMP by transmitting a blank line, or by entering a command that releases the terminal.

5.3 ATTENTIVE TASK

Each terminal has, at most, one attentive task associated with it; the commands affect only that task.

It is the user's responsibility to catalog interactive tasks with the resources needed for each terminal. An interactive task name should consist of 3 CAN code characters followed by the 3 digits of the ID number of the terminal with which it should interface.

For example, a user would like to run a Job Control task on each of eight terminals. (Note: Running Job Control gives the flexibility of running all MODCOMP system processors as well as programs.) You could catalog eight batch tasks, named BAT001 through BAT008, with the resources required to converse with the proper terminal and share files, then enter the following command to initiate the task:

```
>RUN  BAT
```

TMP appends the 3-digit terminal ID when activating the task.

NOTE: The terminals are numbered in decimal so the LINEMONITER statement for that terminal appears in the SYSGEN. The first LINEMONITOR statement assigns the ID number "001" to its terminal, the second LINEMONITOR statement assigns the ID number "002", and so on.

5.4 ERROR MESSAGES

TMP outputs a message in the following format in response to an improperly formatted command:

```
! ERROR  [parameter]    [code]
```

where [parameter] is the number of the parameter in error, if it can be determined, and [code] is an optional 3-character error code.

EXAMPLE

```
! ERROR  Ø  CMD
```

5.5 COMMANDS

The commands recognized by TMP are described in the following pages. The following symbols and notations are used in the syntax descriptions:

[ ]	Brackets indicate either an optional parameter or characters that are not significant.
XXX	Upper-case letters represent items that are to be entered as shown.
xxx	Lower-case letters represent parameters that are defined by the user.

ABORT, KILL

ABNORMALLY TERMINATE THE EXECUTION OF THE ATTENTIVE TASK FOR THIS TERMINAL

These commands cause the attentive task to have its execution and all I/O operations terminated. The ABO command also causes TMP to stop accepting commands from the terminal.

SYNTAX

```
A[BO]
K[IL]
```

ERROR MESSAGES

TNF - Task Not Found
No attentive task is assigned to this terminal.

ASSIGN DEVICE OR LOGICAL FILE TO A LOGICAL FILE OF ATTENTIVE TASK

The ASSIGN Command allows a user to assign a file of the attentive task to another logical file or device. The DEF command sets the default assignment for a device or other logical file.

SYNTAX

```
                        1       2
    ASS           [logfilenam,name-1] ...
    DEF
```

[logfilenam] - Parameter 1 is optional. It is a logical file name in the File Assign Table of the attentive task.

[name-1] - Parameter 2 is required if Parameter 1 is specified. It is a logical file name or device name to which logfilenam should be assigned.

ERROR MESSAGES

TYP - Parameter Type
The specified parameter contains illegal characters.

ASG - Assign Error
The indicated assignment could not be made.

AVA

DISPLAY AVAILABLE PORTS

The AVA Command (a JM overlay) lists all available ports (User ID=Ø) on a multi-user system. AVA uses the PORTINFO REX service to access the User ID information, which is maintained by the MAGIC product.

SYNTAX

```
>AVA
```

EXAMPLE

```
>AVA
 PORT 2
 PORT 7
 PORT 8
```

Multi-user ports 2, 7 and 8 are not in use. All other ports are active.

ADVANCE TO START OF NEXT FILE

The AVFILE Command positions a device to the record immediately following the next end-of-file mark.

SYNTAX

```
                    1   2
     >AVF           lfn [x]
```

lfn - Parameter 1 (required) represents a valid logical file name. If parameter 1 is missing, no operation is performed.

[x] - Parameter 2 (optional) represents the number of file marks to advance. (Default = 1)

ERROR MESSAGES

NAM - Parameter 1 was not a valid logical file name.

EXAMPLE

```
>AVF BI 3
```

AVRECORD

ADVANCE TO THE BEGINNING OF THE NEXT RECORD

The AVRECORD Command positions a logical file to the beginning of the next record (skip one record).

SYNTAX

```
                    1   2
    >AVR          lfn [x]
```

lfn - Parameter 1 (required) represents a valid logical file name. If Parameter 1 is missing, no operation is performed.

[x] - Parameter 2 (optional) represents the number of records to advance. (Default = 1)

ERROR MESSAGES

NAM - Parameter 1 was not a valid logical file name.

EXAMPLE

```
    >AVR BI 3
```

DISPLAY OR CHANGE TERMINAL BAUD RATE

The BAU Command displays and/or changes the baud rate of a terminal connected to a CLASSIC through a 1907 communication device.

WARNING

Baud rates changed with this command must match the terminal hardware switches.

SYNTAX

```
                  1
>BAU            [xxx]
```

[xxx] - Parameter 1 (optional) represents the new baud rate value. If omitted, the current baud rate is displayed.

ERROR MESSAGES

DEVICE NOT 1907
BAU only works on 1907 type communication devices.

DEVICE NOT IN SYSTEM
BAU could not find the attentive terminal listed in the system data structures. This error should never occur.

EXAMPLES

>BAU
The current baud rate is 9600.

>BAU 1200
The current baud rate is 9600. The new baud rate is 1200. The terminal hardware switch must be changed to this rate.

BKFILE

BACKSPACE A FILE ON I/O DEVICE

The BKFILE Command is used on logical files not already at the beginning-of-medium (BOM) position. The service backspaces over physical records until the file mark (EOF) physical record is found, or until the Beginning of Medium position is reached. The next record accessed with a forward device operation will be the EOF record except when the operation results in positioning the device at Beginning of Medium (BOM).

SYNTAX

```
                    1   2
     >BKF           lfn [x]
```

lfn - Parameter 1 (required) represents a valid logical file name. If Parameter 1 is missing, no operation is performed.

[x] - Parameter 2 (optional) represents the number of file marks to backspace. (Default = 1)

ERROR MESSAGES

NAM - Parameter 1 was not a valid logical file name.

EXAMPLE

```
>BKF BI 1
```

BACKSPACE A RECORD OF A LOGICAL FILE

The BKRECORD Command positions a device at the beginning of the previous record (if not already at the Beginning of Medium).

SYNTAX

```
                  1   2
     >BKR         lfn [x]
```

lfn - Parameter 1 (required) represents a valid logical file name. If Parameter 1 is missing, no operation is performed.

[x] - Parameter 2 (optional) represents the number of records to backspace. (Default = 1)

ERROR MESSAGES

NAM - Parameter 1 was not a valid logical file name.

EXAMPLE

```
>BKR BI 1
```

BRO

BROADCAST A MESSAGE TO ALL TERMINALS

The BRO Command (a JM overlay) broadcasts a message to all terminals of a multi-batch system that have a nonzero User ID. This directive uses the PORTINFO REX Service to access the User ID information, which is maintained by the MAGIC product.

SYNTAX

```
                    1
>BRO          message
```

message - Parameter 1 (required) represents an ASCII string.

EXAMPLE

```
>BRO  THERE IS A STAFF METTING AT 2:00 PM TODAY
```

DISPLAY DEVICE INFORMATION

The DEV Command displays basic information about a partition or device.

SYNTAX

```
                  1
>DEV              xxx
```

xxx - Parameter 1 (required) represents a physical partition or device name.

ERROR MESSAGES

CAN - Device name not CAN codable.

IDN - Invalid device name.

EXAMPLES

(The following examples use this Key)

DEV = Parameter 1.
ADDR = Address of logical device table (LDT).
STAT = Device status.
TNA = Physical Transport name.
TCTA = Address of task that has exclusive use of this device.
RSL = Record size limit.

```
>DEV SLS
     DEV   ADDR   STAT   TNA   TCTA   TYPE   EXCLTN   RSL
     SLS   18F3   0080   H00   105B   1801            256

>DEV MT1
     DEV   ADDR   STAT   TNA   TCTA   TYPE   EXCLTN   RSL
     MT1   131A   0000   MT1   130F   0002             0

>DEV SL4
     DEV   ADDR   STAT   TNA   TCTA   TYPE   EXCLTN   RSL
     SL4   1A39   0080   H00   10BB   A001            256

>DEV SMC
     DEV   ADDR   STAT   TNA   TCTA   TYPE   EXCLTN   RSL
     SMC   1AFC   0080   H00   10BB   A001            256
```

```
>DEV SSM
     DEV    ADDR    STAT    TNA    TCTA    TYPE    EXCLTN    RSL
     SSM    1B09    0000    H00    10BB    0001              0

>DEV SBM
     DEV    ADDR    STAT    TNA    TCTA    TYPE    EXCLTN    RSL
     SBM    1B30    0000    H00    10BB    0001              0
```

DISPLAY ANY FILE MANAGER FILES THAT ARE OFFLINE FOR THIS TASK

In TMP the FOL Command displays the File Manager files that are offline for the attentive task.

SYNTAX

```
>FOL
```

EXAMPLES

```
>FOL
     No File Manager files are offline.

>FOL
     LFN = SCA IS OFFLINE, TASK = BAT020
     LFN = BI IS OFFLINE, TASK = BAT020
```

HOLD EXECUTION OF THE ATTENTIVE TASK FOR THIS TERMINAL

The HOLD Command causes the attentive task's execution to hold. The execution can be resumed by either the RESUME or RELEASE command, or by the OC package.

SYNTAX

H[OLD]

ERROR MESSAGES

TNF - Task Not Found
No attentive task is assigned to this terminal.

NORMALIZE DEVICE

The HOME Command normalizes or homes a specified logical file (therefore a physical device). If the particular device cannot perform such an operation, no movement occurs. This service uses the MAX IV REX call, HOME,#8. Refer to the MAX IV EXECUTIVE (REX) SERVICES, System Guide Manual for more information.

SYNTAX

```
                    1
>HOME          lfn1 [....lfnx]
```

lfn1 [....lfnx] - Parameter 1 (at least one is required) represents a valid logical file to be homed.

ERROR MESSAGES

@NAM - Invalid logical file name, or no logical file name specified.

@LOK - The disc wrote a lock error. Check lock switches on disc.

@TAS - No attentive task.

@ERR - No operation was performed. Unidentified error.

EXAMPLE

```
>HOM BO, BI, LO
```

The physical devices associated with logical files BO, BI and LO are normalized.

MSG

SEND A MESSAGE TO A SPECIFIED PORT

The MSG Command (a JM overlay) sends a message to a specified terminal of a multi-batch system. MSG uses the PORTINFO REX Service to access the User ID information, which is maintained by the MAGIC product.

SYNTAX

```
                    1      2
     >MSG          port# message
```

port# - Parameter 1 (required) represents a port number determined by a terminal's position in a Terminal Control List (TCL).

message - Parameter 2 (required) is an ASCII string. This message is sent to the port that the user specified as Parameter 1.

EXAMPLE

```
     >MSG 6 GOOD MORNING!
          The user of Port 6 will receive the display
          "GOOD MORNING!"
```

CALCULATE THIS NUMERIC EXPRESSION

The NUM Command calculates an arithmetic string from left to right (no precedence). It will add, subtract, multiply, and divide (+, -, *, /). It displays the value in Hex, CAN code and decimal.

```
                    1
>N[UM]              xxxx
```

xxxx - Parameter 1 (required) represents a valid arithmetic expression.

EXAMPLES

```
>NUM   #3221
     Hex Value = 0000 3221   Decimal = 12833
     CAN Code = @   H6

>NUM 9/6
     Remainder in Hex = 0000 0003
     Hex Value = 0000 0001   Decimal = 1
     CAN Code = @   A

>N  @ABC
     Hex Value = 0693 0000   Decimal = 110297088
     CAN Code = @ABC

>N  @ABCDEF
     Hex Value = 0693 19CF   Decimal = 110303694
     CAN Code = @ABC DEF
```

GRANT ACCESS TO THE OC PACKAGE

The OCP Command grants the terminal operator access to the OC package, provided the password is acceptable and no other terminal currently has access to the OC package.

NOTE: When sharing the OC package, certain parameters (such as attentive task name "BIAS" value, and PAUSE breakpoint) may not remain valid. In addition, when a user gains access to OC by using the EM option, and the OC package is in use at another terminal, the message "OC REMOVED FROM THIS DEVICE" is displayed on the terminal from which OC is removed.

SYNTAX

```
                    1        2
     O[CP]      [password] [,EM]
```

password - Parameter 1 is optional. It is the current password consisting of from 0 to 8 ASCII characters.

EM - Parameter 2 is optional. It is a keyword that causes the OC package to be delivered to the requesting terminal immediately (providing the password is acceptable). This parameter should only be used in an emergency situation.

ERROR MESSAGES

USE - OC Package in Use
Some other terminal is using the OC Package.

PWV - Password Violation
- No password has been entered and one is required.
- The wrong password has been entered.
- This terminal is denied access to the OC Package.

SET (OR GET) MY ATTENTIVE TASK'S OPTIONS

These commands are used to set, reset, or display the individual bits of the attentive task's system option word or program option word. Syntax is identical to that of the OC OPT and POP Directives (refer to the MAX IV OPERATOR COMMUNICATIONS System Guide Manual listed in the Preface).

SYNTAX

```
OPT             [ [NO]Ø   ]
POP             [ [NO]1   ]
                [  --  -  ]
                [ [NO]F   ]
                [ [NO]LO  ]
                [ [NO]BO  ]
                [ [NO]SC  ]
                [ [NO]HO  ]
                [ [NO]MA  ]
                [ [NO]GO  ]
                [ [NO]AO  ]
                [ [NO]HL  ]
                [ [NO]DU  ]
```

ERROR MESSAGES

TNF - Task not found
The attentive task could not be found.

TYP - Parameter Type
The specified parameter contains illegal/unrecognized characters.

PAS

MODIFY THE OC PASSWORD (OC DIRECTIVE)

This is an OC Directive. It causes the current OC password to be modified.

SYNTAX

	1
/PAS	[password]

[password] - Parameter 1 is optional. It is the new OC password.

EXAMPLES

/PAS 12345678	Causes the current password to be changed to "12345678".
/PAS	Removes the password (a password is not needed on the OCP Directive).
/PAS,,	Changes the password to eight commas (,,,,,,,,) and does not allow the OCP Directive to be processed since its password will not ever compare.

RESUME THE EXECUTION OF THE ATTENTIVE TASK FOR THIS TERMINAL

These commands (except LET) cause the terminal's attentive task to resume execution. Additionally, all these commands set the specified task variable. The R[ES] command also causes TMP to stop accepting commands from the terminals.

SYNTAX

```
                    1     2
R[ES]           [word[,value]]
REL
LET
```

[word] - Parameter 1 is optional. It is the name of a task variable.

[value] - Parameter 2 is required only if Parameter 1 is specified. It is the decimal or hexadecimal value to be given to the variable. The default value is #8000.

ERROR MESSAGES

TYP - Parameter Type
The specified parameter contains illegal characters.

TNF - Task Not Found
No attentive task is assigned to this terminal.

REW

REWIND LOGICAL FILE

The REW Command positions a device assigned to a specified logical file to the Beginning of Medium (BOM) position if such a position is defined for the device, either physically or logically. This command uses the REX services REWIND #2.

SYNTAX

```
                    1
>REW          lfnl [...lfnx]
```

lfnl [...lfnx] - Parameter 1 (required) represents a valid logical file name.

ERROR MESSAGES

@NAM - Invalid logical file name, or no parameter specified.

@TAS - No attentive task.

@ERR - No operation was performed. Unidentified error.

EXAMPLE

```
>REW BI
```

RETURN OC TASK'S OC FILES TO DEFAULT AND SUSPEND OC TASK.

This is an OC Directive. It returns the OC Task's OC and CO files to their defaults and then suspends the OC task. This OC directive must be used to release the OC package (OC task) so that TMP can again be entered when a break or control "A" is detected.

SYNTAX

/ROC

ASSIGN AN ATTENTIVE TASK TO THIS TERMINAL AND ACTIVATE IT

The RUN Command causes TMP to activate a task concatenated with the three-character terminal ID number. That task becomes the attentive task for the terminal. Additionally, TMP stops accepting commands from the terminal.

If more than one task is currently running under control of TMP at a terminal, the attentive task name may also be switched by this directive.

SYNTAX

```
                    1    2
     RUN          nam [fil]
```

nam - Parameter 1 is the first three characters of a valid task name.

[fil] - Parameter 2 is optional. It is the global logical file from which the task is to be loaded. The default is BM.

ERROR MESSAGES

TYP - Parameter Type
The specified parameter contains illegal characters.

IFN - Illegal File Name
The specified file either does not exist or else is not on a direct access device.

TNF - Task Not Found
The specified task could not be found.

INF - Incompatable Influence Levels
The calling task is of an influence that is numerically higher than the specified task's influence.

NRT - No Room for TCB
Pages are unavailable in the system to allocate this TCB.

OVE - Requested Task is an Overlay
Program was found on the file but the format was incorrect.

NTA - No TCB Available
Not enough transient tasks were included in the SYSGEN to allow activation.

DEESTABLISH AND ABORT THE ATTENTIVE TASK

The STOP Command deestablishes the task specified and then aborts it (the attentive task no longer executes).

WARNING!

Parameter 1 must be entered. If it is a batch task, the TCB unabortable bit is reset and autostart no longer occurs.

SYNTAX

```
                 1
STO              nam
```

nam - Parameter 1 is required. It identifies the task to be stopped.

ERROR MESSAGES

TYP - Parameter Type
Parameter 1 has not been entered or contains illegal characters.

INF - Influence Level
Influence level of TMP is insufficient to deestablish and abort the task.

TNF - Task not found

SYS

DISPLAY SYSTEM START-UP MESSAGE AND PAGES AVAILABLE

The SYS Command displays:

- The revision of the Operating System.
- The date of the current SYSGEN.
- The system start-up message.
- The number of pages available.
- The number of pages allocatable ever.

SYNTAX

```
>SYS
```

EXAMPLE

```
>SYS

*I00  09/01/84 MAX IV SYSTEM
NUMBER OF PAGES: AVAILABLE: 1071    ALLOCATALBE EVER: 1719
```

DISPLAYS WHERE LOGICAL FILES ARE ASSIGNED

The TAS Command displays all or selected logical file assignments. It displays the current assignment, file partition index, default assignment, current file-to-file, and default file-to-file.

SYNTAX

```
                  1
>TAS              lfn1 [...lfnx]
```

lfn1 [...lfnx] - Parameter 1 (required) represents a valid logical file name. If Parameter 1 is missing, all file assignments are displayed to the terminal.

EXAMPLE

```
>TAS  SI,BI,SCA,SCB,V20
 FILE  CDEV  FP-INDEX  DDEV  CUR-FILE-TO-FILE  DEF-FILE-TO-FILE
  SI    A38     0       A38        TY=               TY=
  BI    NO      0       NO         SCA=              SCA=
  SCA   NO      0       NO
  SCB   H11<    0                  V20               V20
  V20   H11<    0
```

NOTE: Logical file V20 is assigned to some File Manager file on transport H11. When the file is assigned to a File Manager directory or data file, TAS only informs which transport the file is located on.

TER

TERMINATE OPERATIONS TO A LOGICAL FILE

The TER Command terminates operations queued to a device by the attentive task.

SYNTAX

```
                    1
>TER          lfn1 [...lfnx]
```

lfn - Parameter 1 (at least one is required) represents a valid logical file name.

ERROR MESSAGES

@NAM - Invalid logical file name.

@TAS - No attentive task.

@ERR - Some unidentified error has occurred and operation was not performed.

EXAMPLE

```
>TER BI
```

WRITE AN END-OF-FILE MARK ON THE LOGICAL DEVICE

The WEOF Command writes and EOF record on the I/O device assigned to the specified logical file.

SYNTAX

```
                    1
>WEO[F]         lfn [...lfnx]
```

lfn1 [...lfnx] - Parameter 1 (required) represents a valid logical file name. If Parameter 1 is missing, no operation is performed.

ERROR MESSAGES

@NAM - Invalid logical file name.

@LOK - Lock error.

@TAS - No attentive task.

@ERR - Some unidentified error has occurred and operation was not performed.

EXAMPLE

```
>WEO BO
```

An End-of-File is written to the I/O device assigned to BO.

WHO

DISPLAYS WHO IS ON THE MULTI-USER SYSTEM

The WHO Command (a JM overlay) displays system User IDs, port numbers and the overlays those users are running for all terminals having a nonzero User ID. This directive uses the PORTINFO REX service to access the User ID information, which is maintained by the MAGIC product.

SYNTAX

```
                    1
>WHO              [port...portx]
```

[port...portx] - Parameter 1 (optional) represents a port. If missing, all ports are checked for activity.

EXAMPLE

```
>WHO
        3               JACK            SED
        4               MARY            TOC
        5               JANE            LIB
       1Ø               KIM             SED
       11               JOE             M4E

>WHO 3
        3               JACK            SED
```

DISPLAYS WHAT THE HOLDUP IS WITH THIS TASK

The WTH Command displays the state of the attentive task. It examines the primary (TCBXST) and secondary (TCBOST) status words and generates messages accordingly. WTH also generates information regarding the task name, I/O which may be offline or stalled, the last loaded overlay of the batch task, and the current PR location.

The following are the processed task states along with their corresponding messages.

SYNTAX

```
>WTH

I/O HOLD:
  TASK taskname
  AWAITING I/O
  I/O: logical file name = device name

OC HOLD:
  TASK taskname
  OPERATOR HOLD

EXECUTING:
  TASK taskname
  PR = #xxxx [MAP ZERO]
```

NOTES: If the CKN bit is set in TCBXST, the "CAN'T KILL NOW" message is displayed along with the other status.

If the task is a batch type task, the last loaded overlay message is displayed as follows:

```
LAST LOADED OVERLAY   overlay name
```

If the task is a batch type task, the following message is displayed:

```
BETWEEN <$JOB><$EOJ>
        <$EOJ><$JOB>
```

The current state of I/O requests is displayed as:

```
I/O: logical file name = device name [OFFLINE]
I/O: logical file name = FILE MGR TRANSPORT tna [OFFLINE]
```

```
STALLED DEVICE device name
```

```
QUEUED:
  TASK task name
  INACTIVE
  SUSPENDED
  PR = #xxxx [MAP ZERO]

NOT FOUND:
  TASK NOT FOUND

MEMORY HOLD:
  TASK task name
  AWAITING MEMORY ALLOCATION

SUSPENDED:
  TASK task name
  SUSPENDED
  PR = #xxxx [MAP ZERO]

ROLLED:
  TASK task name
  ROLLED OUT
  PR = #xxxx [MAP ZERO]
```

EXAMPLE

```
>WTH
     TASK BAT020
     PR = #B309
     I/O: CI = A38
     LAST LOADED SED
```

5.6 OVERLAYS

TMP directives are processed as overlays from the standard disc partition QSM. This chapter provides information required to write custom TMP overlays. As soon as an attention request (CONTROL/A) is recognized, TMP assigns an input/output buffer to that terminal in order to input the command. Since there may be fewer buffers available than TMP terminals, a Buffer Control Block (BCB) is maintained by TMP within its virtual addressing space for each buffer area. The layout of the BCB is described in Appendix G of this document.

When TMP inputs the command into the input/output buffer, it then needs an area into which it can load an overlay to process this command. Since the number of Overlay Process Areas (OPA's) does not have to equal the number of input/output buffers or the number of terminals, an Overlay Control Block (OCB) is used for each OPA. The OCB's and OPA's are maintained by TMP in its virtual addressing space. The layout of the OPA is described in Appendix G of this document.

The additional virtual addressing space is allocated dynamically by TMP. Memory required for overlay control blocks, buffer control blocks, and buffers is allocated during TMP startup. Pages into which overlays are loaded are allocated on an as-needed basis. The total unallocated virtual space required by TMP is a function of the OCBLN, BCBLN, BUFLN, equates and the task variables OVL and BCB. Refer to the earlier section of this chapter entitled "Catalog Options and Considerations."

When overlay processing is complete, both the BCB and OCB are made available for use by the same or other terminals. This also means that the input/output buffer and the OPA are made available.

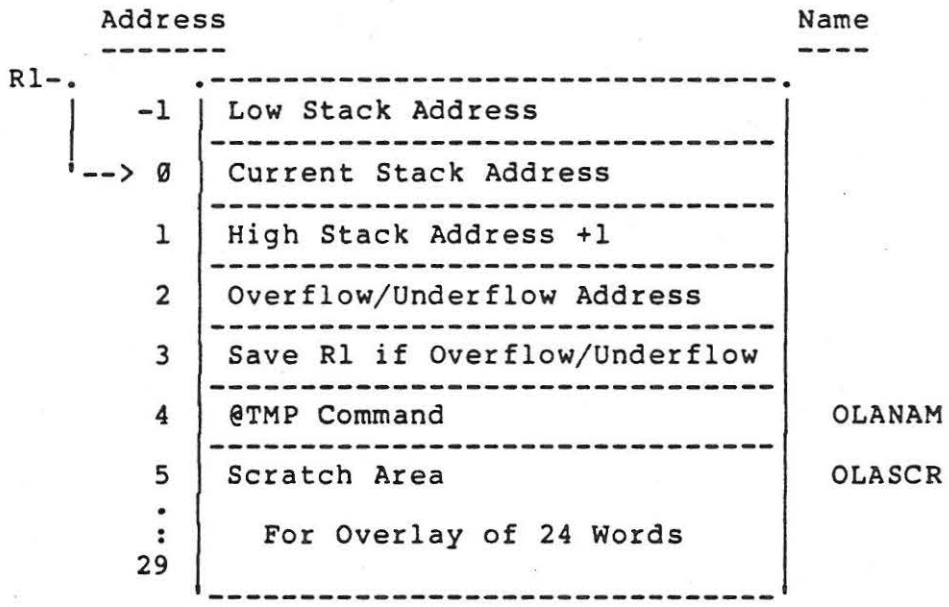
5.6.1 ENTRY TO OVERLAY

Register Contents:

Register	Contents
1	Stack Pointer Table
2	Service Table Address
8	Entry Point to Overlay

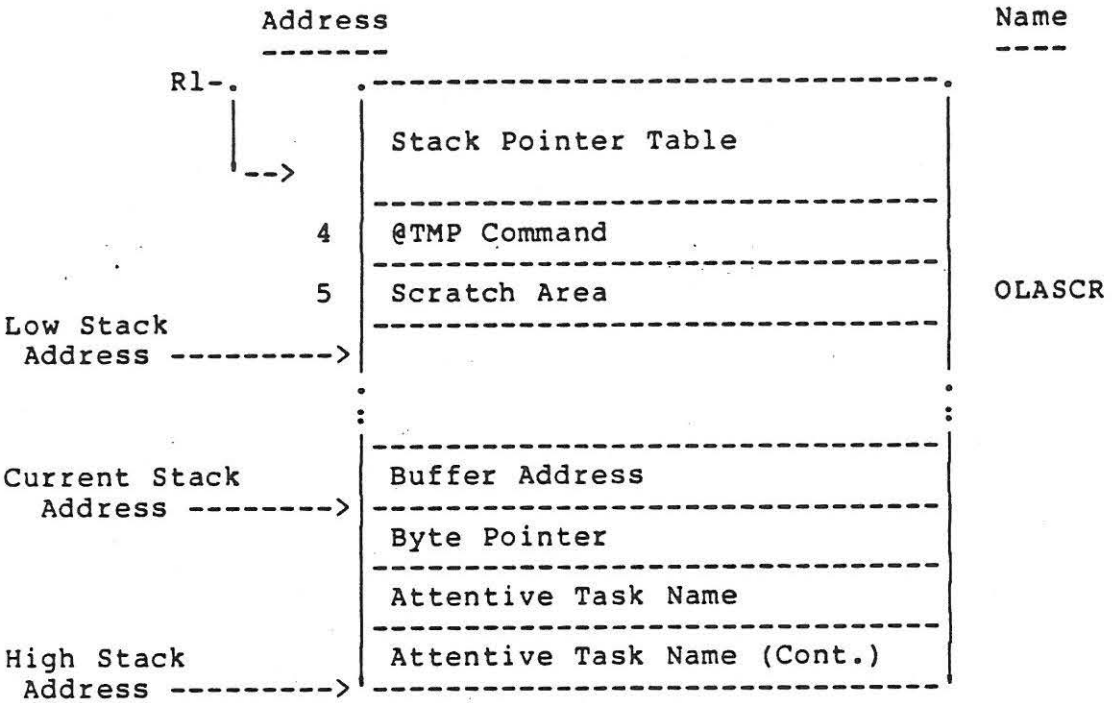
Stack Pointer Table:

Register 1 points to a Stack Pointer Table for use in Push-into-Stack (PSM) and Pull-from-Stack (PLM) instructions. The stack itself is located at an offset of 5 from the address contained in Register 1. In addition, the CAN code TMP command is located at an offset of 4 from the address in Register 1.



5.6.2 STACK

The stack, with a length of 24 words, can be used by the overlay as a scratch area. On entry to the overlay, the stack has the following contents.



The buffer address points to the actual command from the terminal. This buffer is also used by most of the TMP overlay service routines that output messages to the terminal. An exception is the OSWBA routine, which allows the specification of a buffer address in the overlay area itself. The byte pointer points beyond the command and should be used as the starting location for locating parameters. The CAN code attentive task name is the name of the task active at the terminal.

These parameters can be picked up with a PLM instruction.

5.6.3 OVERLAY SERVICE TABLE

This table lists the addresses of common routines that may be accessed from the overlays (refer to the following section on overlay service routines for more information on each routine). Overlays must return to TMP through these routines.

STRUCTURE OF SERVICE TRAP

Address		Name
Ø	RETURN AND READ NEXT COMMAND	OSRET
1	RETURN AND RELEASE TERMINAL	OSREL
2	RETURN AND WRITE BUFFER	OSRWB
3	RETURN, WRITE, AND RELEASE	OSRWL
4	RETURN AND WRITE ERROR MESSAGE	OSRER
5	TMP PARSE ROUTINE	OSPAR
6	WRITE BUFFER	OSWRI
7	CHANGE TCL STATUS	OSCST
8	GET TERMINAL ID	OSTID
9	CHANGE ATTENTIVE TASK NAME	OSCAT
1Ø	GET TCB ADDRESS OF ATTENTIVE TASK	OSTCB
11	SELECT OPERAND MAP	OSSOM
12	GET DEVICE NAMES OF TERMINAL	OSGDN
13	WRITE BUFFER VIA ADDRESS IN R15	OSWBA

5.6.4 OVERLAY SERVICE ROUTINES

The descriptions for these routines follow. Their offset into the Overlay Service Table precede the routine name, and the mnemonic for the offset follows in parentheses. Routines may be called with the Branch and Link Indexed Through Table (BLT) instruction. The return address for all these routines is in Register 8.

OFFSET ROUTINE

0 Return and Read Next Command (OSRET)

An overlay branches to this routine when it has completed processing and wants TMP to read a new command from the terminal.

1 Return and Release Terminal (OSREL)

An overlay branches to this routine when it has completed processing and wants TMP to relinquish control of the terminal.

2 Return and Write Buffer (OSRWB)

An overlay branches to this routine when it has completed processing, wants to output a one-line message to the terminal, and then wants TMP to read a new command. On entry, Register 15 must contain the byte count of the message in the buffer assigned to the terminal.

3 Return, Write Buffer, and Release Terminal (OSRWL)

An overlay branches to this routine when it has completed processing, wants to output a one-line message to the terminal, and then wants TMP to relinquish control of the terminal. On entry, Register 15 must contain the byte count of the message in the buffer assigned to the terminal.

4 Return and Write Error Message (OSRER)

An overlay should branch to this routine when it does not wish to continue processing after detecting an error. Register contents on entry are:

R9: Parameter number in error (binary)
R10: Error code (CAN code)

5 Standard TMP Parse Routine (OSPAR)

This routine helps in parsing commands.

Register contents on entry are:

R7: Parse Table address (refer to Parse Table)
R9: Number of Parameters to be parsed (maximum of 8)
R10: Buffer Address of command

R11: Byte Pointer, where parsing of the command should begin

Register contents on Return:

R9: Number of parameters found, if positive. Two's complement of parameter number if in error
R11: Byte Pointer, after last parameter

Parse Table:

The Parse Table describes the parameters to TMP, and the parameters are returned to the overlay in this table. This table should be constructed in the overlay's scratch area. Parameter types are passed to TMP as follows:

1. Three CAN code characters:

Table Entry Value	Returned in Table
#0001	CAN-Code

2. Six CAN code characters:

Table Entry Value	Returned in Table
Two \ #0002	CAN code of first 3 characters
Words / Any	CAN code of last 3 characters

3. Single Precision numeric:

Table Entry Value	Returned in Table
#0003	Binary value

4. Double Precision numeric:

Table Entry Value	Returned in Table
#0004	Most significant digits
Any	Least significant digits

6 Write Buffer (OSWRI)

Overlays use this routine or the OSWBA routine for performing all output to terminals.

Register Contents on Entry:

R15: Byte count of message in buffer assigned to terminal

Register Contents on Return:

R1: Stack pointer
R2: Service Table Address
R8: Return Address into overlay

All other register contents are indeterminate.

7 Change Terminal Control List (TCL) Status (OSCST)

This routine changes the status in a TCL.

Register Contents on Entry:

R1Ø: New status (upper byte; lower byte ignored).

Register Contents on Return:

R1Ø: Old status/Terminal number.

Registers Destroyed:

R2, R3

8 Get Terminal ID (OSTID)

This routine returns the terminal ID number.

Register Contents on Entry:

None

Register Contents on Return:

R13: ID number

9 Change Attentive Task Name (OSCAT)

This routine changes the task assigned to a terminal.

Register Contents on Entry:

R14 and R15: New attentive task name

Register Contents on Return:

R1Ø: Terminal ID number

1Ø Get TCB Address of Attentive Task (OSTCB)

This routine finds the TCB of the task assigned to a terminal.

Register Contents on Entry:

None

Register Contents on Return:

R1: TCB address.
Ø= TCB not found and Taskmaster is unlocked.
1= TCB found, Taskmaster remains locked out and Z-space will be open and mapped.

R2 through R5: Contents indeterminate

11 Select Operand Map (OSSOM)

This routine selects TMP's operand mask and returns to the main program at the location specified in Register 8. (Note: The BLT instruction cannot be used if TMP's operand map is not already selected.)

Register Contents on Entry:

None

Register Contents on Return:

None

Registers Destroyed: R2

12 Get Device Names of Terminal (OSGDN)

This routine returns the logical device names of the active terminal.

Register Contents on Entry:

None

Register Contents on Return:

R14: Logical device name of output to terminal (CAN-Code)

R15: Logical device name of input to terminal (CAN-Code)

13 Write Buffer via Address in R15 (OSWBA)

Overlays use this routine or the OSWRI routine for performing all output to terminals.

Register Contents on Entry:

R15: Buffer address where message is to be printed from. The carriage control is provided by the user in the beginning of the buffer. The maximum length of a message is 82 characters, and the message must be terminated with a Ø (zero).

Register Contents on Return:

R1: Stack pointer
R2: Service Table Address
R8: Return Address into overlay

All other registers are indeterminate.

5.7 NAMING OVERLAYS

TMP commands are either one or three characters long. The overlay name must consist of the command name followed by three colons in the former case or with one colon in the latter case.

EXAMPLES

COMMAND	OVERLAY NAME
R	R:::
RES	RES:

To guarantee the proper operation of TMP, all overlays should make use of TMP's Overlay Service Routines (OSR) for doing I/O. Overlays are normally allowed to run to completion. An OPA will be deallocated as soon as the overlay requests TMP to type its final response to the terminal. (If an overlay needs to return more than one line of output, it will occupy its OPA until it outputs its final line.)

5.8 PROGRAMMING CONSIDERATIONS

TMP overlays are limited to a length of #200(0-#1FF). If the overlay is larger than #200(0-#1FF), it must be divided into separate overlays. Each overlay can load the next overlay with the use of the following:

```
LDMD,R14      OVNAM      R14-R15 CAN CODE OVERLAY NAME
LDI,R13       @SM        INDICATE SM FILE
LDI,R12       START      ADDR. OF WORD 0 OF THIS OVERLAY
LDI,R8        #2D        REX LOVER SERVICE NUMBER
REX,#32
```

NOTE: When naming secondary overlays, do not use a colon (:) as the fourth character or TMP will recognize it as a primary overlay.

TMP overlays should be placed in partition QSM with a minimum of the following resources:

```
FILE SM
PECULIAR PRIVILEGED
PECULIAR RELOCATABLE
PECULIAR RIOCOMPLETE
OVERLAY (NAME)
```

NOTE: A TMP overlay must not be cataloged with a bias using the BIAS Directive.

CHAPTER 6
EVENT LOGGING

This Chapter describes the Event Logging feature of the MAX IV Operating System. After an overview and a discussion of file use, SYSGEN considerations are discussed. Next, the Event Logging Report task's execution is described.

6.1 OVERVIEW

The MAX IV Event Logging mechanism provides the MAX IV user with an optional capability to capture, record and produce reports on the occurence of selected system, hardware, or user-defined events. Each event selected for logging is monitored by MAX IV and, when the event occurs, a record of the event is created and logged on a user specified file by MAX IV. Each record of logged events contains standard fields of information that identify the type of event and its time of occurence as well as variable fields of information that contain event-specific information. Events may also be formatted to an optional user-definable formatted messages file.

Monitored events may be classified into four basic categories:

- Task Operations

  Events included in the task operations category are task activations, task exits, and task aborts. By monitoring these events, users may obtain information on the performance of their tasks as well as develop profiles of system activity. Information about system performance and the demands imposed on the system allow users to optimize their tasks to meet these demands and improve performance.

- Memory Malfunctions

  Solid state memory malfunctions may be detected on the CLASSIC 78xy series of computers and logged by the Event Logging Package. Logging of individual parity errors, unrecoverable parity errors and high error rates allow users to identify component failures as well as monitor error rates to identify potential problems before they become critical.

- Device Failures

  Device failures include spurious, unsolicited interrupts as well as device off-line messages produced by MAX IV. Again, a profile of these events can point to certain hardware or operational problems and allow them to be corrected before they become critical.

- User-Defined Events

  Using the Event Logging REX service users may define their own events.

In order to use the Event Logging mechanism, the user must be aware of its four basic components:

- Its SYSGEN statements
- Its REX service
- Its OC Directives
- Its associated Event Logging Report task

SYSGEN statements and the Event Logging Report task are described later in this chapter. For information about REX Services refer to the MAX IV EXECUTIVE (REX) SERVICES System Guide Manual listed in the Preface. For information about the OC Directives refer to the MAX IV OPERATOR COMMUNICATIONS System Guide Manual listed in the Preface.

In addition, it is important to understand the manner in which the Event Logging data files are used by the Event Logging mechanism and the Event Logging Report task.

6.2 USAGE OF DATA FILES

The Event Logging mechanism allows the user to specify one or two logical files to be used by the Event Logging mechanism and the Event Logging Report task. These Global Logical files may be assigned to any disc or magnetic tape. There are two modes of operations: the Cyclic mode and the Alternating mode.

In the Cyclic mode of operation the Event Logging mechanism stores data on one file and, when the file is full, rewinds the file and begins storing data at the beginning of the file. The user may optionally specify that Event Logging stop when the file is full. In either case, activation of the Event Logging Report task produces a report that reports on the data records from the beginning of the file to the most recently written record.

The Cyclic mode of operation is most useful when logging data for only certain periods of time and producing reports on this data.

In the Alternating mode of operation, the Event Logging mechanism:

1. Fills one data file and automatically switches to an alternate data file to continue storing data.
2. When the first file is full, the Event Logging Report task is automatically activated to report on the contents of the first file.

3. When the second data file is full, the Event Logging mechanism switches back to the first file to store data and activates the Event Logging Report task to report on the contents of the second file.

This Alternating mode of data storage and reporting continues until terminated with the LOG OC Directive.

The Alternating mode allows users to continuously log and report on system events. To safeguard the integrity of the data logged, the Event Logging mechanism does not alternate data files unless the data in the target file has been processed by the Event Logging Report task. This may result in the loss of logging data, but ensures that data in the log files is accurate. To avoid this condition, users should ensure that their data files are large enough to store data during high logging activity periods or should log data to removable media such as magnetic tape that can be removed and processed at a later time.

6.3 SYSGEN STATEMENTS AND OC DIRECTIVE

Up to four SYSGEN statements are used to configure the Event Logging mechanism into the system. These statements are:

EVENTLOG
LOGPOINT
LOGFORMAT
SERVICE

These statements are described in detail in the MAX IV SYSGEN System Guide manual listed in the Preface. The LOG and EVE OC Directives are documented in the MAX IV OPERATOR COMMUNICATIONS System Guide Manual and are used to control the execution of the Event Logging mechanism and the Event Logging Report task. Both the SYSGEN statements and the OC directives allow the user to specify the Event Logging configuration. Event Logging parameters that can be defined at SYSGEN time or modified during system operation with the LOG OC Directive are described below.

6.3.1 EVENT LOGGING OPERATION

The entire Event Logging mechanism may be enabled or disabled when it is configured into MAX IV. At SYSGEN time the user may specify whether or not Event Logging is to be enabled at system initialization. Event Logging may also be enabled or disabled while the system is operating by use of the LOG OC Directive.

6.3.2 STORAGE MODE

As previously described, data may be logged in either Cyclic or Alternating mode. These parameters may be defined at SYSGEN time and modified at run time with the LOG OC Directive.

6.3.3 EVENT LOGGING EXECUTION

Execution of Event Logging is controlled by the RES and SUS parameters of the EVENTLOG SYSGEN statement or the LOG OC Directive. These parameters work in conjunction with the data storage mode parameters and control the execution of Event Logging when a data file becomes full. If the execution parameter is RESUME (RES) and the data storage mode is Alternating mode, the Event Logging mechanism continues execution on a data file full condition unless the alternate data file is also full. In this case, Event Logging will be resumed upon completion of execution of the Event Logging Report Task. If the execution parameter is RESUME and the data storage mode is Cyclic mode, Event Logging rewinds its data file and continues execution when its data file becomes full.

If the value of the execution parameter is SUSPEND (SUS), the Event Logging mechanism is suspended when a data file becomes full regardless of the data storage mode. Execution may be resumed with the LOG OC Directive.

6.3.4 OPERATOR NOTIFICATION

As the Event Logging mechanism is executing, it optionally notifies the operator of certain conditions. Notification may be enabled or disabled with the EVENTLOG SYSGEN parameter or the LOG OC Directive. Notification messages are described below:

MESSAGE	MEANING
EVENT LOGGING FILES SWITCHED	The user has specified Alternating mode and the Event Logging mechanism has switched to the alternate data file.
EVENT LOGGING FILE CYCLED	The user has specified Cyclic mode and the Event Logging mechanism has rewound the file and begun storing data at the beginning of the file.
EVENT LOGGING FILE FULL	The user has specified Cyclic mode and the Event Logging mechanism has rewound the file and begun storing data at the beginning of the file.
BOTH EVENT LOGGING FILES FULL	The user has specified Alternating mode and both data files are full. Event logging has been disabled until the Event Logging Report task processes the logged data or until the data file status has been reset by the LOG OC directive. Either write out a report or purge the buffer.

EVENT LOGGING SUSPENDED	Event Logging has been suspended because of a data file full condition. Either write out a report or purge the buffer.
ACTIVATING EVENT LOGGING REPORT WRITER	The Event Logging Report task has been automatically activated by the Event Logging mechanism.
EVENT LOGGING REPORTER ACTIVATE ERROR	The Event Logging mechanism has attempted to activate the Event Logging Report task and an error has occurred. The operator may determine the cause of the error by using the LOG OC Directive and interpreting the contents of the 'RAC' Parameter.
EVENT LOGGING I/O ERROR: UFT STATUS=#XXXX	While attempting to write the current logging buffer (which was full) or issuing a TASSIGN REX on the logging filename, an I/O error indication was returned in the UFT status. Control is transferred to the X-task to do other work. The I/O operation is retried later until a different error or successful retry occurs. Refer to the MAX IV BASIC I/O SYSTEM System Guide Manual (listed in the Preface) for the meaning of the error status printed.

6.3.5 EVENT LOGGING FILES

Several Global Logical Files are used by the Event Logging mechanism and the Event Logging Report task during their operation. The names of these files may be specified at SYSGEN time with the EVENTLOG SYSGEN statement and modified at execution time with the LOG OC Directive. It should be noted when using the LOG OC directive that dynamic changes of files should not be made between different device types. The particular files that may be specified are:

- The name of the cyclic data file
- The names of the two alternating data files
- The name of the Event Logging format file used by the Event Logging Report task to format output messages
- The name of the file in which the Event Logging Report task is located
- The name of the file on which immediate formatting of events is to occur.

6.3.6 LOGGING NODES

Each event logged by the Event Logging mechanism requires the temporary use of a system data structure known as a "logging node". The number of nodes available to Event Logging is specified at SYSGEN time by the user. The default number of logging nodes is 20. If an event to be logged occurs and no logging node is available, the logging information is lost. Systems requiring a high level of logging should have a larger number of nodes specified. Each logging node contains logging entry-specific data words. Refer to Appendix H. The number of logging entry-specific data words possible in the node defaults to 16, however, the EVENTLOG SYSGEN statement has a parameter that allows the user to increase this value.

6.3.7 LOG POINTS

As mentioned above, the Event Logging mechanism allows the user to optionally capture and record certain system events. The events that may be logged include:

- Task activations
- Task aborts
- Task exits
- Unsolicited hardware interrupts
- Individual memory parity errors
- Pages of memory that have been taken off-line because of unrecoverable memory errors
- Excessive memory parity error rates,
- Device off-line conditions
- Events that are requested to be logged by the EVELOG REX service.

6.3.8 IMMEDIATE FORMATTING OF LOGGING EVENTS

During SYSGEN the user can request inclusion of an optional immediate formatting of events package. If this package is requested then an additional SYSGEN statement, LOGFORMAT, can be supplied. The parameters of this statement are identical to those of the LOGPOINTS statement. If the immediate formatting mechanism is included along with the LOGFORMAT statement then immediate formatting of logging events to the immediate formatting report file occurs. If the mechanism is requested, but the LOGFORMAT statement is not included, then the package is configured but not enabled. It can be enabled for all events or selected system-defined events by the LOG OC Directive.

NOTE: Immediate formatting of logging events to slow devices can cause event logging nodes to be deleted and, in turn, cause logging information to be lost.

6.4 EVENT LOGGING REPORT TASK

The Event Logging Report task reads the data from the log file, formats it according to the format file, and prints a report of each event in chronological order. The report may be generated automatically when a log file fills (if alternating log files have been defined at SYSGEN). To obtain the report upon demand assign and position all files (see the following FILES Section) and execute the report program from Job Control as follows:

```
$EXE EVEREP,SM,AO
```

EVEREP issues a REX, EVELOG to request that the current logging buffer (possibly containing several logging nodes) be "flushed" (written to the appropriate logging file). This is done in case a report is requested upon demand by manually activating EVEREP. Logging must be enabled for the flush to take place. If logging is disabled, the flush request is ignored and nodes currently in the logging buffer remain there until a node is logged that forces a flush due to lack of room in the logging buffer.

NOTE: EVELOG must be specified as a service name in the SERVICE statement of the SYSGEN in order to utilize the flushing feature. If the service is not included, EVEREP will not issue the flush request.

The format file may be created by the user or the default file supplied by MODCOMP is used. To create a format file, the user must use the formatting conventions defined in the Appendix H. The format records must be uncompressed source records.

To generate an Event Logging report task from source, assemble it under M4A, link edit it using M4EDIT, and catalog it using TOC.

The Event Logging report task should be cataloged as a medium to high priority task if it is to be activated automatically. This allows it to report on the entire file before the alternate file fills up.

6.4.1 FILES

File assignments for the Event Logging Report task should be made as follows:

```
CO   Informational Messages
LO   Report Listing
DO   Diagnostic Output
```

If the AO option is used then the following additional assignments should be made:

```
BI   Event Logging File
SI   Formatting File
```

6.4.2 OPTIONS

The Event Logging Report has several options:

- AO Assumes BI and SI files have been previously assigned.
- UØ Dump the specific event information portion of the current logical record of the log file if an error exit is taken.
- U1 Dump the specific event information portion of all logical records in the log file.
- U2 Dump the header plus the specific event information from all logical records of the log file.
- U3 Dump all formatting file records.

If the AO option is specified, the Event Logging Report task stops when it detects End-of-File or End-of-Media. Without AO it stops when it detects End-of-File, End-of-Media, or when it catches up with the current log file index.

6.4.3 REPORT DETAIL LINES

Each line of the report contains the following information.

- o Sequence number assigned by the event logging mechanism as the event is logged.
- o Message format number is the file position index of the event message format in the format file.
- o Number of words of event-specific information in the logical record of the log file.
- o Negative time until midnight in hours, minutes, seconds, and ticks.

6.4.4 SPECIFIC EVENT INFORMATION

Memory Parity Error (CLASSIC only):

- o Message ID (Ø2)
- o Entry condition codes (NZOC)
- o Most recent copy of the Plane Status Register (PSR) Ø
- o Erring data word read from memory from PSR1
- o The extended memory EMA from PSR2
- o Memory address of the erring data from PSR3
- o Check bits and error code from PSR4

(Refer to the CLASSIC CENTRAL PROCESSOR Reference Manual listed in the Publications Catalog for more information on Plane Status Registers.)

Bad Memory Page Deleted:

- Actual page number of the bad memory page.

Task Abort:

- Aborting task name
- Overlay name
- Address of the abort
- Program status at the abort
- Why the task aborted
- Who aborted the task
- CPU utilization in hours, minutes, seconds, and ticks

Device Offline:

- Device name
- Reason for being off-line
- Task name (blank if not File Manager logical device)

I/O Retried Successfully:

- Retry count
- REX number of operation
- Device Position Index where failure occurred (sector/track/cylinder if disc)
- Transport name
- Task name
- Status of controller at time of last failure
- Device-dependent Code

Task Activation:

- Task name
- Global load module file name
- Execution priority
- Limit priority

Task Exit:

- o Task name
- o Overlay name
- o Why the task exited
- o Who made the task exit
- o CPU utilization in hours, minutes, seconds, and ticks

Unsolicited Interrupt:

- o The interrupt level and a flag indicating whether it was a service interrupt or data interrupt.

NOTE: The retry count definition for the I/O Retried Successfully event is handler dependent. Refer to the MAX IV INPUT/OUTPUT Reference Manual for further discussion.

6.4.5 INFORMATION MESSAGES

The following information messages may appear on the operators' console:

EVENT LOGGING REPORT WRITER F.Ø date

Cause - report writer task activation

EVENT LOGGING REPORT WRITER EXITING

Cause - Reading from AO, log file is not alternating, and End-of-Media detected or End-of-File detected.

6.4.6 ERROR MESSAGES

Abnormal termination of the Event Logging Report Writer causes one of the following messages to print:

NO EVENT LOGGING CONFIGURED

Cause - Event Logging was not defined at SYSGEN with the EVENTLOG and LOGPOINT statements.

NO EVENT LOGGING FORMAT FILE CONFIGURED

Cause - Format file was not specified at SYSGEN.

UNEXPECTED END-OF-FILE ON FORMAT FILE

Cause - Format file has no records.

FORMAT NUMBER OUT OF RANGE

Cause - No format record of that number appears in the format file.

NO EVENT LOGGING DATA TO REPORT

Cause - Log file is empty.

REPORT NOT POSSIBLE FROM CYCLED NON-DISC LOG FILE

Cause - Log file is not on a disc.

INVALID FORMAT CHARACTER

Cause - Termination in invalid place or character not in format table.

PRINT LINE OVERFLOW DURING FORMATTING

Cause - Print line is longer than 94 characters (excluding terminator byte).

FORMAT C CONVERSION ERROR

Cause - Illegal CAN code character.

CHAPTER 7
INTERTASK COMMUNICATION SERVICE

This Chapter describes the MAX IV Intertask Communication (ITC) Service. The processes of setting up server definitions, of establishing Virtual Circuits (VC), of sending and receiving messages is described in detail. The ITC Operator Communication Directives are identified and their usage explained. The ITC REX Services are presented with ASSEMBLY language calling sequences, parameters, error conditions, and explanations of returned values.

7.1 OVERVIEW

The MAX IV Intertask Communications Service (ITC) provides the ability for a task to send a message to another task. Messages are variable-length collections of data to be transmitted between the tasks. The message contents are completely up to the communicating tasks.

The service is based on a mechanism in which messages are constrained to flow along distinct full-duplex communication paths. Each path, known as a Virtual Circuit (VC), connects two communicating tasks. A VC must first be opened by agreement between the two tasks, then both tasks may send and get messages to and from each other on that VC. A task may have many VCs open between itself and other tasks, and may even have several distinct VCs open between itself and a single other task.

Each task using ITC has a message queue. When a message is sent on a VC, the message is added to the end of the destination task's queue. (All messages sent on VCs to the same task end up queued chronologically to that task.) A task may, at any time, get the first message on its queue or the first message on a particular VC. If no messages are on a task's queue or on a specified VC, the task can wait for a message (the task is suspended until a message arrives). By waiting for messages, tasks may use ITC for synchronization as well as communication.

ITC also provides a buffering service. Messages are copied out of the server's buffer and into a dynamically allocated system buffer when they are sent, and copied back into the receiver's buffer when they are received.

The amount of buffering that is available is limited by a Flow Control Mechanism on a per-VC basis. Flow can also be selectively switched off and on by the VCENABLE and VCDISABLE services.

There are additional task control features that enable automatic task activation, resumption, and the disposition of messages and VCs after task exits or aborts.

In addition to the communication service, MAX IV ITC also provides an insertion capability. This is the ability to write a third task that can be inserted between two communicating tasks such that they are unaware of the presence of the third task. Also, the VCLINK

feature enables such a third task to drop out of the communication path by joining two of its own VCs so that it no longer takes part in communication.

7.2 USING ITC

Using the Intertask Communications Service is discussed in the following sections:

- Open Virtual Circuit
- Sending Messages
- Receiving Messages
- Flow Control

7.2.1 OPEN VIRTUAL CIRCUIT

When a task wants to initiate a conversation with another task it must first open a Virtual Circuit to that task. In the call to the VCOPEN service of ITC the user specifies only the logical name of the process with which communication is desired. Because opening a VC often represents invocation of a service or resource, this name is called a Server Name or Resource Name. A task that is to receive VCOPENs must be defined as a server. Server names are ASCII character strings. They are not task names, but are mapped into task names by the VCOPEN service. (Refer to the section on SERVER DEFINITIONS.)

VCOPEN creates the Virtual Circuit to the specified server and returns to the caller a VC number. The task then uses this VC number to identify the conversation when sending and receiving messages.

If the caller generates an error when invoking VCOPEN or any other ITC service, there are basically three Error Action options available:

- No Action simply returns the error status to the caller
- Report Errors causes an error message to be reported to Event Logging and the error status returned to the caller
- Report and Abort, the default logs the error message and then aborts the calling task with appropriate reason codes.

The VCOPEN causes a message to be sent to the task referenced by the Server Name. Before data may be exchanged on this Virtual Circuit the server must either accept or reject the call using VCACCEPT or VCCLOSE respectively. If the server's message arrival action is defined as "activate," then the VCOPEN causes an activation of the server if it is not already active. When conversation on the Virtual Circuit is finished, either party can close it with a VCCLOSE.

7.2.2 SENDING MESSAGES

Once a VC is established between two tasks, either task may send the other a message on that VC. The caller invokes the VCSEND service specifying the VC on which the message is to be sent and the contents of the message. The contents of the message may reside in discontiguous buffers in the server's space. The server specifies how many buffers there are and then gives the address and length (in words) of each buffer. The original location of the message in several discontiguous buffers is completely invisible to the receiver of the message. The caller can indicate an odd byte length message by setting the Ignore-last-byte option on the send.

When sending messages the server is subject to the flow control mechanism as described below.

When sending messages, ITC copies the caller's buffer(s) to a system buffer before returning from the call. The caller is then free to modify the original buffer without affecting the message sent. The system-buffered message is contiguous in the system receive buffer even if it were copied from discontiguous buffers. Whenever the sender indicates the message is an odd byte length, ITC clears the last byte of the received message to a null.

All buffers are verified for the appropriate access rights in the caller's map. If an error is found, an error status is returned to the caller. Debugged programs may use the Noverify option to bypass the buffer verification.

The same Error Action options (no action, report errors, report and abort) available in VCOPEN, are also available in the SEND service.

7.2.3 RECEIVE MESSAGES

A task may receive incoming messages by invoking the VCGET and MSGGET facilities of ITC. The receiver can request information about odd byte length messages by selecting the Ignore-last-byte option on the MSGGET or VCGET call. When this option is selected by the receiver and the sender indicated an odd byte length message (by setting the Ignore-last-byte option on the VCSEND) an odd byte status value is returned to the receiver.

If a message from a particular VC is desired, the receiver may invoke the VCGET service, specifying the VC and buffer into which the message will be copied. If messages from any VC are to be received, the receiver may invoke the MSGGET service, specifying only the receiving buffer. In this case, the number of the VC from which the message (if any) was received is returned to the caller.

When the queued message is not the result of a SEND by the other party, then VCGET and MSGGET return appropriate qualifying information. An additional option to VCGET and MSGGET allows the caller to specify that only control messages are to be returned.

7.2.3.1 No Message Actions

If no messages are available when MSGGET is invoked, or no messages are available on the specified VC when VCGET is invoked, four "no message actions" are available: quick, wait, suspend, and exit.

- Quick - The quick "no message action" returns to the caller immediately with or without a message.

- Wait - The wait "no message action", the default, does not return to the caller without a message. If no message is immediately available, the task is suspended and only the arrival of a message resumes the task.

- Suspend - If no message is immediately available, the task is suspended. The task is resumed by the arrival of a message or by an event such as a resume timer, resume interrupt, REX resume, resume on I/O complete or OC will resume the task without a message. This is useful for tasks that must wait for both messages and other events that cause their resumption.

- Exit - The exit "no message action" causes the task to exit if no message is immediately available. Subsequent arrival of a message can cause the task to be re-activated, if the Activate option is selected as the message arrival action for the server. (Refer to the following section called Task Activation.)

7.2.3.2 Message Overflow Options

When a message is received with the VCGET and MSGGET facilities, the number of words in the message copied into the receiving buffer is returned to the caller. Two options are available for handling messages that are longer than the specified receiving buffer, No-flush and Flush.

- No-flush, the default, leaves the remaining overflow of the message pending pending on the VC. The caller is returned a "more data" (status > Ø). Subsequent VCGET and MSGGET from that VC will get the next portion of the message that fits in the receiving buffer. Thus, a task can intentionally receive messages a few words at a time.

- The Flush option specifies that a message that is too long for the specified buffer is an error. If Flush is specified, any message overflow is discarded and the caller is returned a "message too long" error status (status < Ø).

The same Error Action options (no action, report errors, report and abort) available in the SEND service are also available with these facilities.

7.2.4 FLOW CONTROL

There are two aspects to the flow control mechanism:

- Congestion control
- User control

Congestion control is specified by the user in VCOPEN as the number of messages nodes. If a SEND results in the limit of message nodes being exceeded, the message is not sent and the caller is returned an "insufficient window size" status. If, as a result of a SEND, the window closes (available message nodes is equal to Ø), subsequent SENDs will be returned a status of "window closed." ITC will post a Control Message to the user to provide notification when the flow window has opened.

The second aspect of Flow Control, user control, allows users to invoke the VCENABLE and VCDISABLE routines to selectively start or stop the reception of messages on a given VC by MSGGET. A VCGET will still be able to receive a message.

A Type Word is received when using the VCGET and MSGGET facilities. This word contains the type of information being conveyed. Types include VCOPEN request, VCCLOSE request, DATA and VCACCEPT. See REX MSGGET for detailed types.

7.3 INCLUDING ITC IN A MAX IV OPERATING SYSTEM

Including ITC in a MAX IV Operating System is discussed in the following sections.

- SYSGEN requirements
- Task requirements
- Server requirements

7.3.1 SYSGEN REQUIREMENTS

To include ITC in a MAX IV Operating System, specify the following statement in the System Structure Block (part 1 of the SYSGEN).

```
                          1       2          3
INTERTASK          [pages][,sysdws][,MAP[PED]]
```

Parameter 1 is optional. It refers to the maximum amount of actual memory (in pages) to be used at any given time by ITC. The default is 200. This upper limit prevents ITC from absorbing all of actual memory under adverse conditions such as a runaway task. A status indicating insufficent resources will be returned on any call to ITC until resources are again available.

Parameter 2 (optional) refers to the default window size to be given to tasks not specifying a window size. The acceptable range is from 1 to 127. The default window size is 8 message nodes.

Parameter 3 is specified only if ITC is to run in a map other than MAP Ø. Refer to Non-MAP Ø Resident Code, a section in the Hardware Requirements and Utilization chapter in this manual for a complete explanation of how to run a Non-MAP Ø Resident ITC.

7.3.2 TASK REQUIREMENTS

A permanent data structure is built by ITC for each task that uses it. This data structure is called a "port." For server tasks, the port is built when the server is defined. When the server task is activated, the port is linked into its TCB as an extension. The port for other tasks using ITC is built when the task is activated. The port for a server task remains in the system after the task exits, but a user port is deleted. A port contains information concerning server task activation or resumption, message queues, message and VC disposition on server task exits or aborts, and VCs.

7.3.3 SERVER DEFINITIONS

The main function of the server definition is to associate a server name with a task name. Servers are defined in two ways, either as Permanent Server Definitions or as Transient Server Definitions. Permanent Server Definitions are included in the SYSGEN. Transient Server Definitions are maintained as a source file processed by the ITC routines, SDI and SDF.

The processing routines SDI and SDF read server definitions from the global logical file name SDF. These Transient Server Definition statements can be prepared and maintained by using the Source Editor (SED). The SDI and SDF tasks are discussed in the section on Transient Server Definitions (later in this Chapter).

The server task has several options available. They are specified on the PORT statement and are described below.

7.3.3.1 Task Activation

Servers can be defined in the ITC server directory and its message data structures can exist without the server task itself being active or established in memory. In fact many servers want to be activated only when a message is sent to them. The Activate option can be specified in the option parameters of the PORT statement. If this option is specified, the name of the file in which the server task is cataloged and its activation priority must also be specified. When a message is first sent to a server so specified, the server task will be activated. The server then usually invokes VCACCEPT in order to complete the connect protocol.

7.3.3.2 Task Resumption

A server task that is not marked activate or a task that is already active will be resumed by ITC. For instance, if a task has suspended by using the Wait option on a VC or MSGGET, then when a message is sent to it the ITC resumes the task. If the task is already active, secondary resume is set.

7.3.3.3 Task Exits and Aborts

Server definitions always survive the server task's exits or aborts. Whether or not the open VCs are closed and the pending messages are discarded depends on the "Save VCs on Exit" and "Save VCs on Abort" options. If an option is not specified in the server definition, a VCCLOSE is generated for each open VC with a reason code of abort or exit and all corresponding messages in both directions are discarded.

It must be noted that there is an interaction between the Task Activation option and Exit and Abort options. The Activate option is removed once the server is activated by the arrival of a message. However, a server that was defined with the Activate option can simply exit expecting to be re-activated by the next message or VCOPEN.

When a task not defined in a server definition exits or aborts, its open VCs are closed with the appropriate reason code and all messages in both directions are discarded.

7.3.3.4 Permanent Server Definitions

A permanent server is defined in the system by including the following statement in the System Structure Block, part 1, of the SYSGEN.

SYNTAX

	1	2	3	4	5	6
SERVER	server-name	task-name	[A]	[B]	[C]	[D]

server-name - Parameter 1 is a string of from 1 to 40 ASCII characters terminated by a comma. The following characters cannot be used: blanks, control (non-printing), plus-sign(+), minus-sign(-), equal sign(=), dollar sign($), hexadecimal sign(#), or slash(/).

task-name - Parameter 2 is a MAX IV task name.

A,B,C,D - Parameters 3 through 6 are up to four optional words that contain the user extension of the definition. If any of these words are omitted, a zero value is supplied.

Each task defined by a SERVER statement in the SYSGEN may have a PORT statement. This statement describes:

- Task load file
- Priority
- Maximum number of Virtual Circuits
- Whether a task is activated on message arrival

- o Whether or not VCs and messages are saved or destroyed on exits or aborts

Only one port is defined for each task even if the task is specified by more than one server name. Therefore, for a given task, it is only necessary to specify task information once. This is done by including the following statement in the System Structure Block, part 1, of the SYSGEN.

SYNTAX

```
                    1     2   3    4      5
     PORT      task-name pri lmf maxvcs options
```

task-name - Parameter 1 is a MAX IV task name defined in a SERVER statement.

pri - Parameter 2 is the activation priority to use for the task if no priority was cataloged for the task, and it is not established or prescheduled. (Refer to the REX ACTIVATE Service in the MAX IV EXECUTIVE (REX) SERVICES System Guide Manual listed in the Preface.)

lmf - Parameter 3 is the global load module file for the task.

maxvcs - Parameter 4 is the maximum number of Virtual Circuits that the task can have open at any one time.

options - Parameter 5 may be any combination of:

AMA: Message arrival action is activated. If not specified, a resume occurs.

SOA: Save task's VCs and messages if task aborts, otherwise close VCs and delete messages.

SOE: Save task's VCs and messages if task exits, otherwise close VCs and delete messages.

If the PORT statement is specified, then parameter 2 (task-name) must be specified. If any of the other parameters are missing, they are defaulted. If a PORT statement is not given for a task mentioned in a SERVER statement, all parameters for that task are defaulted. Standard defaults are:

Parameter	Default
pri	255
lmf	LM
maxvcs	4
options	none

SYSGEN Error Messages from SERVER or PORT Statements:

MESSAGE	MEANING
ILLEGAL CHARACTER IN SERVER NAME	Illegal character ("+", "-", "=", "$", "#", "/", or nonprinting) in server name.
NO SERVER NAME SPECIFIED	Server name is required in all SERVER statements.
SERVER NAME TOO LONG	Server name may be up to 40 characters.
TASK NAME NOT SPECIFIED IN SERVER STATEMENT	Task name is required in all SERVER statements.
TASK NAME NOT SPECIFIED IN PORT STATEMENT	Task name is required in all PORT statements.
TOO MANY VC'S REQUESTED IN PORT STATEMENT	More than 255 VC's were requested.
ILLEGAL NUMBER OF VC'S REQUESTED IN PORT STATEMENT	Zero or fewer VC's were requested.

7.3.3.5 Transient Server Definitions

Transient Servers may be initialized in a system in two ways:

- SDI, the server definition initialization task and SDF, the file read task.
- $SVRDEF, a Job Control procedure

SDI may be prescheduled, activated by another task, or OC. Servers may be added at any time. The definitions are the same format as the SYSGEN SERVER and PORT statements. Asterisks may be used for comment lines; blank lines may be added for readability.

SDI builds the data structures required by ITC. SDF reads the definitions from a file that can be maintained by SED. The records can be in compressed form. SDI accesses SDF using ITC which requires a permanent server definition for the server name used by either SDI:

```
PORT      SDF,,SM,,AMA
SERVER    SDF.READ,SDF
```

Or by SDF:

```
PORT      SDI,,SM,,AMA
SERVER    SDF.WRITE,SDI
```

To initialize the transient servers during system startup, enter these statements in the System Structure Block (Part 1) of the SYSGEN:

```
PORT      SDI,,SM,,AMA
SERVER    SDF.INIT,SDI
```

Add sufficient I/O and X task logfile assignments to define and position logical file SDF or include them in SDF resources. Task SDF reads file SDF until an EOF, an EOM or a read error and send the server definitions to SDI using ITC.

During normal system operation, either SDI or SDF may be activated to commence server initialization. The biggest difference is that if SDF is executed as a batch overlay, it uses a local file, whereas if SDI is activated, SDF uses a global file.

To add transient servers after system startup, simply assign and position logical file SDF to the desired definitions and then execute SDI or SDF by a REX activate or OC:

```
/SDI/EXE,,SM
```

Example of Transient Server Definitions:

```
*
* A LIST OF TRANSIENTS FOR MACHINE "A"
*
  PORT   ABC,200,SM,18,SOE
  PORT   XYZ,200,,11
*
  SERVER TASK-NUMBER-1,ABC,#1234,#8,#1,#A
  SERVER TASK-NUMBER-2,XYZ,#1234,,#8
  SERVER TASK-NUMBER-3,ABC,#1234
```

NOTE: In the SDF file, place PORT statements for a task before SERVER statements using that task. Otherwise, the SERVER statement defines a default port, which could result in an error when the corresponding PORT statement is used.

To define servers from a batch task, the user may use the Job Control $SVRDEF statement.

SYNTAX

```
                  1
$SVRDEF        name
```

name - Parameter 1 is a file-name cataloged on the current USL. If "name" is omitted, SI is read from its current file position as the server definition file. $SVRDEF then overlays the batch task with task SDF.

The actual procedure is:

```
$PROD SVRDEF,,SM
$IFP %1,2
$MOVE SI SDF
$AVR CI 2
$ASS SDF USL
$POS %1 SDF
$EXE SDF %2
$ASS SDF SDF
$END SVRDEF
```

7.3.3.6 SDI and SDF Tasks

Both SDI and SDF are divided into two parts. The first part of task SDI opens a Virtual Circuit to task SDF. If accepted, the second part of SDI issues VCGETs, to receive the SERVER and PORT statements from SDF using ITC. SDI then syntactically validates the records and if error free, builds the ITC data structures. Messages are displayed to SDI's CO file to indicate SDI start, successful end, or abnormal end and errors. If executed with the NOBO option, SDI skips the build phase and allows an edit check on the source statements. If executed with the LO option, records are output to the LO file as they are processed. Under normal operation, SDI permits up to 10 incorrectly parsed statements and then exits. This is to allow for the possibility of improperly positioned files. The user may change this default by setting the task variable "ERR" when cataloging the SDI task with the Task Overlay Cataloger.

Task SDF's first part consists of a MSGGET to receive the VCOPEN from SDI, and a VCACCEPT to establish the path. SDF's second part issues read requests against logical file SDF using L$CMR to process possible compressed source. Messages are displayed to SDF's CO file to indicate SDF start, errors and end. File SDF is not repositioned before or after processing. All messages have a data-time stamp.

Custom user routines may be linked into SDI and SDF. Between the first and second parts of each task, a call is made to allow user processing after the Virtual Circuit has been established, but before any data transfers or structures have been built. For instance, this allows user security processing or file positioning. These calls use the FORTRAN calling sequence:

In SDI the call is:

```
                LDI,10          RETADR
                BLM,8           USRSDI
                DFC             1
                DFC             VC
RETADR          EQU             $
```

In SDF the call is:

```
                LDI,10          RETADR
                BLM,8           USRSDF
                DFC             2
                DFC             SERVER
                DFC             VC
RETADR          EQU             $
```

SERVER is the address of the server name used to involve SDF. The name is up to 40 characters terminated by a null.

If the user does not supply routines for USRSDI or USRSDF, the default routines are used by the link edit. The default routines are BRX,10. The object modules of SDI and SDF are found in AL4. USRSDI and USRSDF are contained on ALB.

7.4 ITC OPERATOR COMMUNICATIONS (OC) DIRECTIVES

The MAX IV ITC OC Directives are a set of three directives that can be used by the system operator to obtain information about ITC. Each task that uses ITC must have a port structure. The port structure contains information that is necessary to ITC and is task specific. The PRT Directive outputs the information stored in the port structures to the operators' console. Each port also acts as a queue head for messages sent to the port's task. The MES Directive prints out a list of information about messages queued to a port. Finally, an association exists between server names and tasks. The server names that a task may have can be printed out by using the SVR Directive.

The following descriptions provide more information about these three directives:

7.4.1 MES - MESSAGE

MES is an OC Directive that can be used to print out information about messages queued to one or all ports. Information returned consists of the task name for each port visited as well as a list containing the VC number of a message, a message size in words, and a message type for each message queued to the port. Message information are returned in chronological order for each port. An error return is made if ITC is not in the system.

SYNTAX

```
                1
/task-name/[DIS] MES
           [TYP]
```

task-name - Parameter 1 specifies the name of a particular port's task. It is only specified if message information is requested of only one port. An error return is made if a port does not exist for the specified task.

7.4.2 PRT - PORT INFORMATION

PRT is an OC Directive that can be used to print out information for one or all ports in the port table. Information returned consists of the name, load module file, and activation priority for the port's task, as well as the options, maximum number of VCs permitted, type, and current number of open VCs for each port requested. An error return is made if ITC is not in the system.

SYNTAX

```
        1
/task-name/[DIS] PRT
           [TYP]
```

task-name - Parameter 1 specifies the name of a particular port's task. It is only specified if port information is requested of only one port. If defaulted, information for all ports is given. An error message is printed if the task specified has no port structure.

7.4.3 SVR - SERVERS

SVR is an OC Directive that can be used to print out a list of server names along with their associated task names for one or all tasks in the system. An error return is made if the ITC is not in the system.

SYNTAX

```
        1
/task-name/[DIS] SVR
           [TYP]
```

task-name - Parameter 1 specifies the name of a particular task for which a list of server names associated with the task are requested. If defaulted, then information for all tasks is given. A message is printed if the task specified has no server name.

7.5 INTERTASK COMMUNICATIONS SERVICES (#42)

The following ITC Services are discussed:

- VCOPEN
- VCACCEPT
- VCSEND
- VCCLOSE
- VCGET
- MSGGET
- VCENABLE
- VCDISABLE
- VCLINK
- VCINFO
- SDINFO
- PRTDEF
- SVRDEF
- VCRESET

VIRTUAL CIRCUIT OPEN REQUEST

Call Name:	VCOPEN	Call Number:	#0042 (REX,#32)
Option Names:	ITNOABORT	Option Value:	#0001
	ITNOREPORT		#0002
	ITQUICK		#0040

SERVICE PERFORMED

This service sends an open request control message to the task identified by the server named in the calling sequence. The arrival action specified by the server definition is performed.

CALLING PARAMETERS

R8: Call number and option bits

Bits 4-15 = #042

R10: Address of server name string in the caller's operand space.

R11: Options:

Bit 9: Return mode
=0 The return is to await a message from the server.
=1 Return immediately after the open message is sent. Option name = ITQUICK.

Bit 14: Report mode
=0 ITC errors are reported to Event Logging.
=1 ITC errors are not reported to Event Logging. Option name = ITNOREPORT.

Bit 15: Abort mode
=0 ITC errors do abort the calling task.
=1 ITC errors do not abort the calling task. Option name = ITNOABORT.

[R13:] Window size in message nodes. If not specified, the SYSGEN default is used. One message node can handle up to 248 words of message.

R14: =0 (Required)

R15: =0 (Required)

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return. Status in R3 is 0.
'0000	Abnormal return. Status detail in R3.

REGISTERS CHANGED UPON NORMAL RETURN

R3: Status:
0 = Success. The operation was performed as requested for a Quick Mode VCOPEN. For a Wait mode operation, the server task replied with a VCACCEPT.

R12: The VC identifier to use in future references to this logical communications path. It is meaningful only on this side of the VC.

REGISTERS CHANGED UPON ABNORMAL RETURN

R3: Status:
6 = The server task replied to the "open" request with a VCCLOSE with a reason code. A VCGET or MSGGET may be used to receive the reason code.
-5 = No memory available for a port structure.
-8 = Calling task has no free VC slot available.
-9 = Server task has no free VC slot available.
-16 = Activation failure of server task. Message remains in system.
-17 = Invalid activation load module file. Message remains in system.
-20 = Server named in open request not found.
-41 = Invalid window size (greater than 127 or less than zero).

USAGE EXAMPLE

Send an "open" request to SERVER1 in Quick mode. The calling task will look for a response later, before the first VCSEND is attempted.

```
        LDI,R8          VCOPEN          CALL NUMBER
        LDI,R10         SVNAM           SERVER NAME STRING
        LDI,R11         ITQUICK         QUICK MODE RETURN
        ZRR,R13                         USE THE SYSGEN DEFAULT WINDOW SIZE
        SURD,R14,R14                    SET TO ZERO FOR VCOPEN
        REX,MAXIV                       REQUEST OPENING OF VC
        HNR,ERROR                       TRANSFER IF NOT SUCESSFUL
        STM,R12         VCNUM           SAVE VC NUMBER
         -
         -
         -
SVNAM   DFC             "SERVER1 "      SERVER NAME STRING
VCNUM   DFC             0               VC IDENTIFIER
```

ACCEPT VIRTUAL CIRCUIT OPEN REQUEST

Call Name:	VCACCEPT	Call Number:	#0142 (REX,#32)
Option Names:	ITNOABORT	Option Value:	#0001
	ITNOREPORT		#0002

SERVICE PERFORMED

This service confirms the opening of a Virtual Circuit between two tasks by establishing the logical communications path and sending an "accept" control message to the task requesting the VC.

CALLING PARAMETERS

R8: Call number:

Bits 4-15 =#142

R11: Options:

Bit 14: Return mode
=0 ITC errors are reported to Event Logging.
=1 ITC errors are not reported to Event Logging. Option name = ITNOREPORT.

Bit 15: Abort mode
=0 ITC errors abort the calling task.
=1 ITC errors do not abort the calling task. Option name = ITNOABORT.

R12: The VC identifier to use in future references to this logical communications path. It is meaningful only on this side of the VC.

[R13:] Window size in message nodes. If not specified, the SYSGEN default is used. One message node can handle up to 248 words of message.

R14: =0 (Required)

R15: =0 (Required)

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return. Status in R3 is 0.
'0000	Abnormal return. Status detail in R3.

REGISTERS CHANGED UPON NORMAL RETURN

R3: Status:
Ø = Success. Operation performed as requested.

REGISTERS CHANGED UPON ABNORMAL RETURN

R3: Status:
-41 = Invalid window size. Greater than 127 or less than zero.
-3 = Invalid VC. The VC does not belong to the calling task or does not exist.

USAGE EXAMPLE

By using a MSGGET, the calling task had received a VCOPEN request from another task. The calling task wishes to accept the request and establish communications.

```
      LDI,R8          VCACCEPT    CALL NUMBER
      ZRR,R11                     USE REPORT AND ABORT DEFAULTS
      LDM,R12         VCNUM       VC NUMBER FROM PREVIOUS MSGGET
      LDI,R13         2Ø          WINDOW SIZE OF 2Ø MESSAGE NODES
      SURD,R14,R14                SET TO ZERO FOR VCACCEPT
      REX,MAXIV                   ACCEPT THE VC OPEN REQUEST
       -
      HNR,ERROR
       -
VCNUM DFC             Ø           VC IDENTIFIER SAVE FROM MSGGET
```

SEND A MESSAGE ON A VIRTUAL CIRCUIT

Call Name:	VCSEND	Call Number:	#0242 (REX,MAXIV)
Option Names:	ITMDATA	Option Value:	#1000
	ITNOABORT		#0001
	ITNOREPORT		#0002
	ITILBYTE		#0004
	ITNOVERIFY		#0100
	ITQDATA		#2000

SERVICE PERFORMED

The VCSEND service sends a message on a specified Virtual Circuit. "M" and "Q" data indicators may be specified in the Options parameter.

The contents of the message can reside in discontinguous buffers in the server's space. Up to 32,768 buffer and length pairs can be specified. If the total message length exceeds the space available in the current window, control returns to the caller with status information. None of the message is sent unless there is sufficient window to send the entire message.

Buffer length is specified in number of words. Messages of odd byte length can be signalled to the receiver with the ignore-last-byte option, ITILBYTE. When this option is used, ITC clears the last byte in the received message to null and makes odd byte status available to the receiver.

CALLING PARAMETERS

R8: Call number:

Bits 4-15: =#242

R11: Options:

Bit 2: "Q" data indicator.
=0 Do not assert "Q" data indicator.
=1 Assert "Q" data indicator in received message type. Option name = ITQDATA.

Bit 3: "M" data indicator
=0 Do not assert "M" data indicator.
=1 Assert "M" data indicator in received message type. Option name = ITMDATA.

Bit 7: Buffer verification mode
=0 Verify buffers for access rights.
=1 Do not verify buffers for access rights. Option name = ITNOVERIFY.

Bit 13: Ignore-last-byte
=Ø Do not ignore the last byte. It contains valid data.
=1 Ignore last byte, because the message is of odd byte length. Option name = ITILBYTE.

Bit 14: Report mode
=Ø Report ITC errors to Event Logging.
=1 Do not report ITC errors to Event Logging. Option name = ITNOREPORT.

Bit 15: Abort mode
=Ø ITC errors do abort the calling task.
=1 ITC errors do not abort the calling task. Option name = ITNOABORT.

R12: The VC identifier.

R14: Address of a list of buffer and length pairs.

R15: The number of buffer and length pairs.

CONDITION CODES UPON RETURN

NZOC	Condition
'1ØØØ	Normal return. Status in R3 is Ø.
'ØØØØ	Abnormal return. Status detail in R3.

REGISTERS CHANGED UPON NORMAL RETURN

R3: Status:
Ø = Success. Operation performed as requested.

REGISTERS CHANGED UPON ABNORMAL RETURN

R3: Status:
5 = VC no longer connected. The other side has issued a VCCLOSE.
-3 = Invalid VC. The VC does not belong to the calling task or does not exist.
-1Ø = Protocol violation. THe VC is in a state other than "data".
-16 = Activation failure of server task. Message remains in system.
-25 = Invalid buffer address, access rights or length.
-26 = Invalid number of segments.
-43 = Insufficient window exists to send the entire message. None of the message has been sent.
-44 = Window is closed. None of the message has been sent. A "FLOW CONTROL WINDOW OPEN" message will be received when the window has at least one message node available.

USAGE EXAMPLE

The calling task has already accepted or received an accept on a VC. Now data is to be exchanged on the VC.

```
        LDI,R8          VCSEND          CALL NUMBER
        ZRR,R11                         USE REPORT AND ABORT DEFAULTS
        LDM,R12         VCNUM           VIRTUAL CIRCUIT IDENTIFIER
        LDI,R14         SEGLIS          ADDRESS OF SEGMENT LIST
        LDM,R15         NUMSEG          NUMBER OF SEGMENTS
        REX,MAXIV                       SEND THE MESSAGE
          -
        HNR,ERROR                       ERROR?  IF SO, GO PROCESS
          -
SEGLIS  DFC             MSGBF1          ADDRESS OF MESSAGE BUFFER 1
        DFC             LTHBF1          LENGTH OF BUFFER 1 IN WORDS
        DFC             MSGBF2          ADDRESS OF MESSAGE BUFFER 2
        DFC             LTHBF2          LENGTH OF BUFFER 2 IN WORDS

NUMSEG  DFC             2               NUMBER OF SEGMENTS IN SEGMENT LIST

VCNUM   DFC             Ø               VC IDENTIFIER FROM VCOPEN OR MSGGET
```

CLOSE A VIRTUAL CIRTUAL

Call Name: VCCLOSE Call Number: #0342 (REX,#32)

Option Names: ITNOABORT Option Value: #0001
ITNOREPORT #0002

SERVICE PERFORMED

This service closes the logical communications path between the two participating tasks. The task issuing the VCCLOSE will not be able to make subsequent accesses to the VC. If the other task continues to send, it will receive a status of "slots not connected", and the messages will be ignored. The VCCLOSE control message may include a 16-bit reason code. Certain codes are reserved. (See PRTDEF R11 options for those codes.)

A VCCLOSE removes all data messages in both directions. This implies that a task willing to do a VCCLOSE or ready to exit without the Save Messages and VCs on Exit option should wait for a reply from the other task indicating that all data has been processed. Otherwise, the data messages that are still in the receive queue are discarded by the VCCLOSE processing. Note that the other task's message remains in its receive queue until the time that it dequeues the "close" control message.

CALLING PARAMETERS

R8: Call number:

Bits 4-15: =#342

R11: Options:

Bit 14: Report mode
=Ø ITC errors are reported to Event Logging.
=1 ITC errors are not reported to Event Logging. Option name = ITNOREPORT.

Bit 15: Abort mode
=Ø ITC errors abort the calling task.
=1 ITC errors do not abort the calling task. Option name = ITNOABORT.

R12: The VC identifier.

R13: A 16-bit reason code for the closing of the VC.

R14: =Ø (Required)

R15: =Ø (Required)

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return. Status in R3 is 0.
'0000	Abnormal return. Status detail in R3.

REGISTERS CHANGED UPON NORMAL RETURN

R3: Status:
0 = Success. Operation performed as requested.

REGISTERS CHANGED UPON ABNORMAL RETURN

R3: Status:
-3 = Invalid VC. The VC does not belong to the calling task or does not exist.

USAGE EXAMPLE

The calling task desires to terminate communications on a specific VC.

```
        LDI,R8          VCCLOSE     CALL NUMBER
        LDI,R11         ITNOABORT   USE NO ABORT OPTION
        LDM,R12         VCNUM       VC IDENTIFIER
        LDI,R13         #1220       USER SUPPLIED REASON CODE
        SURD,R14,R14                CLEAR R14, R15 FOR CALL
        REX,MAXIV                   CLOSE THE VC
         -
        HNR,ERROR                   ERROR? IF SO, GO PROCESS
         -
VCNUM   DFC             0           VC IDENTIFIER FROM VCOPEN OR
*                                   MSGGET
```

RECEIVE A MESSAGE FROM A SPECIFIC VIRTUAL CIRCUIT

Call Name: VCGET Call Number: #0442 (REX,MAXIV)

Option Names: See MSGGET

SERVICE PERFORMED

The VCGET service returns a message from a specific VC. The message returned will be the oldest data message in the queue, or the first control message if a control message is present.

Whenever the sender selects the ignore-last-byte option (sets ITILBYTE), ITC clears the last byte of the received message to a null. The receiver can request odd byte status by selecting the ITILBYTE option on the VCGET call. When the receiver queries ITC with ITILBYTE (Bit 13) set and the sender set ITILBYTE in the option word of the VCSEND, an odd byte status is returned to the receiver in Register 3.

CALLING PARAMETERS

R8: Call number:
Bits 0-3: =0
Bits 4-15: Service call number =#442 and call name = VCGET.

R11: See MSGGET.

R12: The VC identifier.

R14: The address of the receive buffer.

R15: The length of the receive buffer in words.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return. Status in R3 is 0.
'0000	Abnormal return. Status detail in R3.

REGISTERS CHANGED UPON NORMAL RETURN

See MSGGET.

REGISTERS CHANGED UPON ABNORMAL RETURN

See MSGGET.

R3: Status:
-3 = Invalid VC. The VC does not belong to the calling task or does not exist.

USAGE EXAMPLE

The calling task wants to wait until a message is received from a specific Virtual Circuit.

```
LDI,R8          VCGET           CALL NUMBER
LDI,R11         ITNOABORT!ITNOREPORT
LDM,R12         VCNUM           VC ID FROM A VCOPEN OR MSGGET
LDI,R14         MSGBFR          ADDRESS OF RECEIVE BUFFER
LDI,R15         BFRLTH          LENGTH OF BUFFER IN WORDS
REX,MAXIV                       VCGET, WAIT, NO ABORT OR REPORT
  -
HNR,ERROR                       ERROR?  IF SO, GO PROCESS IT
```

GET A MESSAGE FROM ANY VIRTUAL CIRCUIT

Call Name:	MSGGET	Call Number:	#0542 (REX,MAXIV)
Option Names:	ITCONTROL	Option Value:	#0010
	ITEXIT		#0060
	ITFLUSH		#0200
	ITQUICK		#0040
	ITNOABORT		#0001
	ITNOREPORT		#0002
	ITILBYTE		#0004
	ITNOVERIFY		#0100
	ITSUSPEND		#0020

SERVICE PERFORMED

The MSGGET service returns a message from any VC. The message returned will be the oldest data message in the queue, or the first control message if a control message is printed.

Whenever the sender selects the ignore-last-byte option (sets ITILBYTE), ITC clears the last byte of the received message to a null. The receiver can request odd byte status by selecting the ITILBYTE option on the MSGGET call. When the receiver queries ITC with ITILBYTE (Bit 13) set and the sender set ITILBYTE in the option word of the VCSEND, an odd byte status is returned to the receiver in Register 3.

CALLING PARAMETERS

R8: Call number:
- Bits 0-3: =0
- Bits 4-15: Service call number =#542 and call name = MSGGET.

R11: Options:
- Bit 6: Flush Message Overflow
  - =0 Save the words in excess of the receive buffer size for the next ITC receive service call.
  - =1 Discard the words in excess of the receive buffer size. Option name = ITFLUSH.
- Bit 7: Buffer Verification mode
  - =0 Verify buffers for access rights.
  - =1 Do not verify buffers for access rights. Option name = ITNOVERIFY.
- Bits 9-10: Return mode
  - ='00 Wait for a message to be sent from the other task.
  - ='01 Suspend until a message or other resume occurs. Option name = ITSUSPEND.

='10 Return immediately with or without a message. Option name = ITQUICK.
='11 Exit if no messages are present. Option name = ITEXIT.

Bit 11: Control Message option
=0 Return data and control messages.
=1 Return only control messages. Option name = ITCONTROL.

Bit 13: Ignore-last-byte
=0 Do not indicate odd byte length in R3 status.
=1 Set the appropriate status in R3 if the last byte is not valid data. Option name = ITILBYTE.

Bit 14: Report mode
=0 Report ITC errors to Event Logging.
=1 Do not report ITC errors to Event Logging. Option name = ITNOREPORT.

Bit 15: Abort mode
=0 Abort the calling task for ITC errors.
=1 Do not abort the calling task for ITC errors. Option name = ITNOABORT.

R12: The VC identifier.

R14: The address of the receive buffer.

R15: The length of the receive buffer in Words.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return. Status in R3 is 0.
'0000	Abnormal return. Status detail in R3.

REGISTERS CHANGED UPON NORMAL RETURN

R3: Status:
0 = Success. Operation performed as requested.

R12: The VC identifier corresponding to the received message.

R13: The type of received message.
0 = Data message.
1 = Data message with "M" indicator.
2 = Data message with "Q" indicator.
3 = Data message with "M" and "Q" indicators.

The following message types are control message:
4 = Open request. The buffer contains the server name used.
5 = Accept acknowledgement. The receive buffer is unchanged.

6 = Close notification. The buffer contains a 16-bit reason code.
12 = Window open on VC. The receive buffer is unchanged.
13 = Reset notification. The buffer contains a 16-bit reason code.

R15: The length of the received message in words.

REGISTERS CHANGED UPON ABNORMAL RETURN

R3: Status:
7 = Ignore-last-byte option. The receiver should ignore the last byte of the message. This status value is returned only when the MSGGET call asks for odd byte information (ITILBYTE, Bit 13 set) and the sender indicated odd byte length by setting ITILBYTE in the option word of the VCSEND.
2 = No message is available in the receive queue.
1 = Message too long for receive buffer. The other registers are as in the normal return. A subsequent ITC receive service call may be used to receive the rest of the message.
-2 = Invalid buffer address, length or access rights.
-29 = Flush option. The message was too long for the receive buffer and the excess message length was discarded. The other registers are as in the normal return.

USAGE EXAMPLE

The calling task wants to suspend until a message is received, a delay timer resumes the task, an I/O operation completes (resume on I/O complete) or it is resumed by another task or OC. This example assumes that the task was cataloged with a PECULIAR RIOCOMPLETE statement to TOC.

```
        LDI,R8          MSGGET          CALL NUMBER
        LDI,R11         ITNOABORT!ITNOREPORT!ITSUSPEND OPTIONS
        LDI,R14         MSGBFR          ADDRESS OF RECEIVE BUFFER
        LDI,R15         BFRLTH          LENGTH OF BUFFER IN WORDS
        REX,MAXIV                       MSGGET, SUSPEND, NO ABORT OR
*                                       REPORT
        CRI,R3          2               MESSAGE AVAILABLE?
        HZS,NOMSG                       IF NOT, GO CHECK TYPE OF RESUME
        TRR,R3,R3                       DID SOME OTHER ERROR OCCUR?
        HNS,ERROR                       IF SO, GO PROCESS ERROR

GOTMSG  EQU             $               PROCESS MESSAGE WITH STATUS =
*                                       Ø = 1
```

ENABLE A VIRTUAL CIRCUIT

Call Name: VCENABLE Call Number: #Ø642 (REX,#32)

SERVICE PERFORMED

This service sets the state of the Enable Status flag for the VC to allow a MSGGET to receive messages from the VC. A VC is in the "enabled" state unless a VCDISABLE has been performed on it. See VCDISABLE.

CALLING PARAMETERS

R8: Call number:

Bits 4-15: =#642

R12: The VC identifier.

CONDITION CODES UPON RETURN

NZOC	Condition
'1ØØØ	Normal return. Status in R3 is Ø.
'ØØØØ	Abnormal return. Status detail in R3.

REGISTERS CHANGED UPON NORMAL RETURN

R3: Status:
Ø = Success. Operation performed as requested.

REGISTERS CHANGED UPON ABNORMAL RETURN

R3: Status:
-3 = Invalid VC. The VC does not belong to the calling task or does not exist.

USAGE EXAMPLE

A task has previously "disabled" a Virtual Circuit and now wishes to put it into the "enabled" state.

```
      LDI,R8      VCENABLE      CALL NUMBER
      LDM,12      VCNUM         VC IDENTIFIER FROM A VCOPEN
*                                OR MSGGET
      REX,MAXIV                 ENABLE THE VC
       -
      HNR,ERROR                 IF ERRORS, GO PROCESS THEM
```

VCDISABLE

DISABLE A VIRTUAL CIRCUIT

Call Name: VCDISABLE Call Number: #0742 (REX,#32)

SERVICE PERFORMED

This service sets the state of the Enable Status flag for the VC to prevent a MSGGET from receiving messages from the VC. A VC is in the enabled state unless a VCDISABLE has been performed on it. The disabled state keeps a MSGGET from receiving any messages except a CLOSE message from the specified VC. A VCGET receives messages from the specified VC regardless of the state of the Enable Status flag.

CALLING PARAMETERS

R8: Call number:

Bits 4-15: =#742

R12: The VC identifier.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return. Status in R3 is 0.
'0000	Abnormal return. Status detail in R3.

REGISTERS CHANGED UPON NORMAL RETURN

R3: Status:
0 = Success. Operation performed as requested.

REGISTERS CHANGED UPON ABNORMAL RETURN

R3: Status:
-3 = Invalid VC. The VC does not belong to the calling task or does not exist.

USAGE EXAMPLE

A task wants to use a MSGGET to receive messages, but wants to prevent messages from being received from a specific VC.

```
      LDI,R8        VCDISABLE      CALL NUMBER
      LDM,R12       VCNUM          VC IDENTIFIER FROM A VCOPEN
*                                  OR MSGGET
      REX,MAXIV                   DISABLE THE VC
        -
      HNR,ERROR                    IF ERRORS, GO PROCESS THEM
```

LINK TWO VIRTUAL CIRCUITS TOGETHER

Call Name: VCLINK Call Number: #0842 (REX,#32)

SERVICE PERFORMED

In the case where task A has VC1 established to task B and VC2 established to task C, a call to VCLINK by task A connects tasks B and C together and eliminates task A from the logical communications path. Tasks B and C continue to reference the same VC identifiers returned to them during the original VCOPEN service. Messages are forwarded in the appropriate directions and maintain their original ordering. Both VC1 and VC2 must be in the data state. The flow control windows will be as defined by tasks B and C. Note that tasks B and C may continue to execute while the VCLINK is in progress so long as they do not reference the VC that is being linked. In that instance, the task relinquishes until the VCLINK is complete.

CALLING PARAMETERS

R8: Call number:

Bits 4-15: =#842

R12: The VC identifier of one Virtual Circuit.

R13: The VC identifier of the other Virtual Circuit.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return. Status in R3 is 0.
'0000	Abnormal return. Status detail in R3.

REGISTERS CHANGED UPON NORMAL RETURN

R3: Status:
0 = Success. Operation performed as requested.

REGISTERS CHANGED UPON ABNORMAL RETURN

R3: Status:
-3 = Invalid VC. The VC does not belong to the calling task or does not exist.
-4 = Attempt to VCLINK a VC of the calling task that points back to the calling task.
-10 = Protocol violation. The VC is in a state other than "data".

USAGE EXAMPLE

A task has a VC open to each of two other tasks. It now wants to link the two tasks together and be removed from the logical communications path.

```
LDI,R8          VCLINK          CALL NUMBER
LDM,R12         VCNUM1          VC IDENTIFIER TO TASK B
LDM,R13         VCNUM2          VC IDENTIFIER TO TASK A
REX,MAXIV                       LINK THE TWO VCS TOGETHER
  -
HNR,ERROR                       ERROR?  IF SO, GO PROCESS IT
```

OBTAIN INFORMATION ABOUT A SPECIFIC VIRTUAL CIRCUIT

Call Name: VCINFO Call Number: #0942 (REX,#32)

SERVICE PERFORMED

This service returns the present state, flow control windows and the name of the task at the other side of the Virtual Circuit. Flow windows are in the range of 0 to 127 and may be temporarily negative due to CLOSE or FLOW WINDOW OPEN control messages being sent.

CALLING PARAMETERS

R8: Call number:

Bits 4-15: #=942

R12: The VC identifier.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return. Status in R3 is 0.
'0000	Abnormal return. Status detail in R3.

REGISTERS CHANGED UPON NORMAL RETURN

R3: Status:
0 = Success. Operation performed as requested.

R10: Current receive flow control window.

R11: Current transmit flow control window.

R13: The state of the Virtual Circuit on the calling task's side.
0 = Error
1 = Not in use
2 = Open request has been received
3 = Open request has been sent
4 = Data

R14: CAN code characters 1-3 of the other task's name.

R15: CAN code characters 4-6 of the other task's name.

REGISTERS CHANGED UPON ABNORMAL RETURN

R3: Status:
-3 = Invalid VC. The VC does not belong to the calling task or does not exist.

USAGE EXAMPLE

Obtain information about a Virtual Circuit of the calling task.

```
      LDI,R8        VCINFO      CALL NUMBER
      LDM,R12       VCNUM       VC ID FROM MSGGET OR VCOPEN
      REX,MAXIV                 GET INFO ON THIS VC
        -
      HNR,ERROR                 IF ERROR, GO PROCESS IT

VCNUM DFC           Ø           VC IDENTIFIER
```

OBTAIN INFORMATION ABOUT A SERVER DEFINITION

Call Name: SDINFO Call Number: #ØA42 (REX,#32)

SERVICE PERFORMED

This service returns information concerning a server definition and options. The user may optionally request that the user data extension to the server definition be returned.

CALLING PARAMETERS

[R7]: The address of a 4-word user data table or -1.

R8: Call number:

Bits 4-15: =#A42

R1Ø: Address of server name string in the caller's operand space. The string must be terminated by a blank or null byte.

CONDITION CODES UPON RETURN

NZOC	Condition
'1ØØØ	Normal return. Status in R3 is Ø.
'ØØØØ	Abnormal return. Status detail in R3.

REGISTERS CHANGED UPON NORMAL RETURN

R3: Status:
Ø = Success. Operation performed as requested.

R9: The maximum count of VCs that the server may have open simultaneously.

R1Ø: The current count of VCs that the server has open.

R11: The ITC options presently configured for the server task.
Bit Ø: The message and VC disposition upon server task exits.
=Ø Discard messages and close VCs with reason code -1.
=1 Save messages and allow VCs to remain open.

Bit 1: The message and VC disposition upon server task aborts.
=Ø Discard messages and close VCs with reason code -2.
=1 Save messages and allow VCs to remain open.

Bit 3: Server task is to be activated upon message arrival.

R12: The server task's load module logical file name.

R13: The server's activation priority to use if none was specified when the task was cataloged.

R14: CAN code characters 1-3 of the server task's name.

R15: CAN code characters 4-6 of the server task's name.

REGISTERS CHANGED UPON ABNORMAL RETURN

R3: Status:
-35 = Server name not found.

USAGE EXAMPLE

Obtain information concerning "SERVER1", but do not return the optional user data.

```
      LDI,R7        -1          DO NOT RETURN USER DATA
      LDI,R8        SDINFO      CALL NUMBER
      LDI,R1Ø       SERVER      ADDRESS OF SERVER NAME STRING
      REX,MAXIV                 OBTAIN SERVER INFORMATION
       -
      HNR,ERROR                 IF ERROR, GO PROCESS IT
       -
SERVER DFC          "SERVER1 "
```

DEFINE OR REDEFINE A SERVER TASK'S PORT

Call Name:	PRTDEF	Call Number:	#ØB42 (REX,#32)
Option Names:	ITCSOE	Option Value:	#8ØØØ
	ITCSOA		#4ØØØ
	ITCAMA		#1ØØØ

SERVICE PERFORMED

This service defines or redefines the parameters contained in a server's port. The port is the ITC extension to a task's TCB that is present when the task is using ITC. Note that ports defined during the SYSGEN cannot be redefined. Also, the maximum VCs parameter cannot be changed once it has been defined.

Two types of ports exist within ITC. A server port is created for tasks that are defined to be servers. The other type, a user port, is built implicitly for other tasks when they issue a call to ITC. A user port resides in the system only as long as the associated task, and may not be redefined into a server port.

CALLING PARAMETERS

R8: Call number:

Bits 4-15: =#B42

R9: The number of VCs the server may have concurrently open
=Ø To obtain the default of 4 VCs.
=1 Up to 57 VCs may be specified.
(To obtain up to 255 VCs, the port must be defined in the SYSGEN.)

R11: The ITC options to be configured for the server task.
Bit Ø: The message and VC disposition upon server task exits.
=Ø Discard messages and close VCs with reason code -1.
=1 Save messages and allow VCs to remain open. Option name = ITCSOE.
Bit 1: The message and VC disposition upon server task aborts.
=Ø Discard messages and close VCs with reason code -2.
=1 Save messages and allow VCs to remain open. Option name = ITCSOA.

Bit 3: Server task activation upon message arrival.
=Ø Do not activate server task.
=1 Activate server task upon message arrival. Option name = ITCAMA.

R12: The server task's load module logical file name.

R13: The server's activation priority to use if none was specified when the task was cataloged.

R14: CAN code characters 1-3 of the server task's name.

R15: CAN code characters 4-6 of the server task's name.

CONDITION CODES UPON RETURN

NZOC	Condition
'1ØØØ	Normal return. Status in R3 is Ø.
'ØØØØ	Abnormal return. Status detail in R3.

REGISTERS CHANGED UPON NORMAL RETURN

R3: Status:
Ø = Success. Operation performed as requested.

REGISTERS CHANGED UPON ABNORMAL RETURN

R3: Status:
3 = Port parameters for an existing port were successfully redefined.
-13 = Invalid maximum VCs parameter.
-36 = Attempt to redefine a user or SYSGEN port.
-37 = Port is presently being redefined by another task.

USAGE EXAMPLE

Define a port for a server task.

```
      LFM,R8          PRTCAL      CALL PARAMETERS
      REX,MAXIV                   DEFINE PORT
       -
      HNR,ERROR                   IF ERROR, GO PROCESS IT

PRTCAL DFC            PRTDEF      R8:  CALL NUMBER
      DFC             1Ø          R9: MAX VCS OF 1Ø
      DFC             Ø           R1Ø: NOT USED
      DFC             ITCSOE!ITCSOA!ITCAMA  R11: OPTIONS
      DFC             @BM         R12: LOAD MODULE FILE LOGICAL
*                                 NAME
      DFC             2ØØ         R13: USE PRI=2ØØ IF NONE
*                                 WAS CATALOGED BY TOC
      DFC             @SVR        R14: SERVER TASK NAME
      DFC             @TSK        R15: SERVER TASK NAME
```

CREATE A SERVER DEFINITION

Call Name: SVRDEF Call Number: #ØC42 (REX,#32)

SERVICE PERFORMED

This service creates a server definition that is used to map the server name to a task. An optional table of up to four words of user-supplied data may be kept with the definition. Also, any number of server definitions may point to the same task. A server name may have its associated server task name or user data changed, but the definition cannot be removed from the system. Note also that a SYSGEN-defined server definition is considered to be a permanent definition. This is to facilitate MAX IV restart.

CALLING PARAMETERS

[R7:] The address of a 4-word user data table or -1.

R8: Call number:

Bits 4-15: =#C42

R1Ø: Address of server name string in the caller's operand space. The string must be terminated by a blank or null byte.

R14: CAN code 1-3 characters of the server task's name.

R15: CAN code 4-6 characters of the server task's name.

CONDITION CODES UPON RETURN

NZOC	Condition
'1ØØØ	Normal return. Status in R3 is Ø.
'ØØØØ	Abnormal return. Status detail in R3.

REGISTERS CHANGED UPON NORMAL RETURN

R3: Status:
Ø = Success. Operation performed as requested.

REGISTERS CHANGED UPON ABNORMAL RETURN

R3: Status:
4 = Server parameters successfully redefined.
-30 = Attempt to change a permanent definition.
-39 = Invalid character found in server name.

USAGE EXAMPLE

Define a server and initialize the user data.

```
      LDI,R7        USRDAT     ADDRESS OF DATA FOR THIS
*                                DEFINITION
      LDI,R8        SVRDEF     CALL NUMBER
      LDI,R10       SVRNAM     ADDRESS OF SERVER NAME
      LDI,R14       @SVR       SERVER TASK NAME
      LDI,R15       @TSK
      REX,MAXIV                CREATE SERVER DEFINITION
        -
      HNR,ERROR                IF ERROR, GO PROCESS IT

USRDAT DFC          #A000      USER DATA FOR THIS DEFINITION
       DFC          @LM
       DFC          80
       DFC          -1

SVRNAM DFC          "SERVER1 "
```

ISSUE A RESET CONTROL MESSAGE

Call Name: VCRESET Call Number: #ØD42 (REX,#32)

SERVICE PERFORMED

This service issues a "reset" control message on a specific VC. It has no other effect on ITC except to deliver a 16-bit reason code to the receiver at the higher priority of the control message queue.

CALLING PARAMETERS

R8: Call number:

Bits 4-15: =#D42

R12: The VC identifier.

R13: A 16-bit reason code for issuance.

CONDITION CODES UPON RETURN

NZOC	Condition
'1ØØØ	Normal return. Status in R3 is Ø.
'ØØØØ	Abnormal return. Status detail in R3.

REGISTERS CHANGED UPON NORMAL RETURN

R3: Status:
Ø = Success. Operation performed as requested.

REGISTERS CHANGED UPON ABNORMAL RETURN

R3: Status:
-3 = Invalid VC. The VC does not belong to the calling task or does not exist.

USAGE EXAMPLE

The calling task needs to notify the other task of some application specific condition that has occurred.

```
      LDI,R8        VCRESET    CALL NUMBER
      LDM,R12       VCNUM      VC ID FROM MSGGET OR VCOPEN
      LDI,R13       #8ØAB      USER REASON CODE
      REX,MAXIV                ISSUE RESET
        -
      HNR,ERROR                IF ERROR, GO PROCESS IT

VCNUM DFC           Ø          VC IDENTIFIER
```

7.6 ITC USER STATE TRANSITIONS

An ITC user can be in one of four possible states on each Virtual Circuit known to the task:

- Free
- VCOPEN Received
- VCOPEN Sent
- Data

To initiate communication between two tasks, the originator must issue a VCOPEN. An available Virtual Circuit is selected and moved from the "Free" to the "VCOPEN Sent" state. Subsequent VCOPENs to the same server would create additional paths. The server task then issues a MSGGET and will receive the VCOPEN request and Virtual Circuit number. The following table summarizes the call and state combinations.

USER CALL	USER STATES			
	1 FREE	2 RECEIVED VCOPEN	3 SENT VCOPEN	4 DATA
VCOPEN	OK-3	NA	NA	NA
VCACCE	IVC	OK-4 (1)	PV	PV
VCSEND	IVC	PV	PV	OK (2)
VCGET	IVC	OK	OK (3)	OK (4)
VCCLOS	IVC	OK-1	OK-1	OK-1
MSGGET	OK (5)	NA	OK (5)	OK (5)

Table 7-1. ITC Transitions

NOTES:

OK-n - Valid call, "n" is the next state, otherwise there is no state change.

IVC - Invalid Virtual Circuit error.

PV - Protocol violation.

NA - Does not apply (state is not possible)

(1) , - Next state is normally 4 (DATA). However, the originator may have decided to send a VCCLOS and not wait for your VCACCE. Your VCSENDs is accepted, but no error returned until you receive the VCCLOS by a VCGET or MSGGET. None of the messages sent are received.

(2) - Next state is normally 4 (DATA). If the other side of the VC decides to VCCLOS, your VCSENDs is accepted, but not received. No error is returned until the VCCLOS is received by a VCGET or MSGGET.

(3) - If a VCACCE is received, the next state is 4 (DATA). If a VCCLOS is received, the VCOPEN was rejected by the server and the next state is 1 (FREE).

(4) - Next state is normally 4 (DATA). If a VCCLOS is received, the next state is 1 (FREE).

(5) - MSGGET is the call used to receive VCOPENs. Otherwise, MSGGET operates like a VCGET, except that the VC is a returned rather than a supplied parameter.

CHAPTER 8
QUICK LOAD MODULES

This chapter defines the organization, format, and content of the MAX IV Operating System quick load module data structure. The term "quick load module" is used to distinguish the complete program modules of MAX IV from those of MAX III. MAX III program modules require relocation by software, while MAX IV program modules do not. The term "quick load module" is shortened to "load module" in the remainder of this chapter.

8.1 DEFINITIONS

A load module contains:

1. A representation of the instructions and/or operands of a task or task overlay in a form suitable for execution and/or use within the CLASSIC/MAX IV system environment.

2. Directions for:

   a. Assignment and/or use of task resources or overlay resources.

   b. Manipulation and use of the virtual/actual memory space into which the module contents will be loaded and/or will execute.

   c. Definition or redefinition of task variables and/or options.

   d. Loading of the instructions and/or operands.

   e. Definition of module entry point.

A load module is created by one of the following:

- The task and program management facilities of the MAX IV Task/Overlay Cataloger (TOC).

- The MAX IV Operating System Roller Task when the element is configured in the system and a task needs to be rolled-out to permit reallocation of its actual memory to a higher priority task.

All load modules may be loaded by the MAX IV Operating System Module Loader (ML).

A file which convains a set of information identifying individual records of the file and their location within the file is termed a "directoried data file". The totality of all such identification and location information within the file is termed the "file directory".

A load module consists of a named collection of records contained in a directoried data file resident on randomly accessible secondary storage. A load module describing a task must be resident in a globally defined file. A load module describing an overlay may be resident in a globally defined or in a locally defined file.

A load module contains two basic types of records:

1. Module Object Records (MOR's) which contain the representation of the instructions and/or operands of the task or overlay.

2. Load Program Records (LPR's) which contain all information necessary to establish an environment for the module and to load the module for execution or use.

8.2 MODULE OBJECT RECORDS (MORs)

MOR's contain the binary representation of the instructions and/or operands of a task or overlay after all processing by the MAX IV Task/Overlay Cataloger has been completed.

The location of these records and their loading directions are contained on the LPR(s) of the module. The MOR's are designed to be loaded via data-chaining capabilities of the Direct Memory Processor.

A single MOR always begins on an integral sector boundary and always occupies an integral number of sectors. The number of sectors occupied by a single MOR is a function of a variety of factors, including:

1. Characteristics of the physical device on which the medium containing the load module is mounted.

2. Contiguity of the medium space allocated to the load module file and available when the load module is created.

3. The order in which MOR contents are placed in the module.

4. The virtual memory characteristics of the task or overlay.

5. Installation-specified parameters pertaining to overall system performance.

8.3 LOAD PROGRAM RECORDS (LPRs)

LPRs contain the directions which permit the Module Loader (ML) to:

1. Establish the proper environment for the module.
2. Load the module.
3. Prepare for execution or use of module contents.

A single LPR occupies all or part of one sector in the load module file. The loading directions for a single module may require one or more LPR's.

An LPR contains one or more variable-length Load Program Instructions that are to be processed by the ML. All LPR's contain only integral instructions (that is, an instruction is entirely contained in a single LPR). Except for the last LPR of a module, all LPR's must contain at least two instructions (since one of the instructions causes reading of the following LPR when necessary).

8.4 DIRECTORIED DATA FILE CONTAINING LOAD MODULES

The directory of a file containing load modules is organized as a "segmented list" and is processed sequentially. A segmented list consists of records which are one sector in length. The first four words of each record of a segmented list have the following format and content:

WORD	BIT	CONTENT
1	Ø	Set to one if this record is the last of the list. Set to zero if another record follows.
	1-6	Ignored by the ML.
	7	Set to zero if 24 bits are used to describe the FPI of the next record of the list (if any). Set to one if 16 bits are used to describe the FPI of the next record of the list.
	8-15	Contains the high order 8 bits of the FPI of the next record of the list (if any) if Bit 7 set to zero. Otherwise, ignored by the ML.
2		Contains the low order 16 bits of the FPI of the next record of the list (if any).
3		Ignored by the ML.
4		#9FFF.

Individual directory entries of the segmented list have the following format and content:

WORD	BIT	CONTENT
1	Ø	Set to one if this entry is the last entry in the segment. Otherwise, set to zero.
	1	Set to one if entry is "vacant" (contains no valid information due to deletion of an entry without directory reorganization).
	5, 6 8-15	Used by TOC to hold the number of sectors the program occupies. If the program occupies more than 1Ø23 sectors this field is set to zero.
2		Contains the low order 16 bits of the FPI of the first LPR of the load module whose name is contained in words 3 and 4 of the entry.
3		First 3 CAN code characters of load module's name.
4		Last 3 CAN code characters of load module's name.

8.5 LOAD PROGRAM INSTRUCTIONS

As necessary, a single instruction may range in length from 1 word to 96 or 126 words -- depending on the sector size of the medium on which the module is resident.

Regardless of the length of the instruction, the first word contains Ø in Bits Ø and 1 and the operation code of the instruction in Bits 2 through 7. The format and content of the remainder of the instruction is solely dependent on the operation code.

When instructions are processed, loading errors may be encountered as a function of existing environmental conditions. The occurrence of a load error sets the ML-defined bit "LEO" (Load Errors Occurred) but does not halt the loading process. If the ML-defined bit "ODM" (Omit Diagnostic Messages) is not set to '1, the occurrence of a loading error cause the ML to invoke the REX Message service to output a message in the following general form to the "CO" file:

LOAD ERROR AT #nnnn -- ccc

where:

nnnn represents the number of the instruction which was being processed in the load program when the error occurred. This

number is keyed to the sequential load instruction as listed by the MAP Directive of the Task/Overlay Cataloger (TOC).

ccc represents a 3-character reason code. Refer to the Error Detection paragraph of each Load Program Instruction section and to Section 8.6.

Regardless of the type of instruction being processed, a single instruction will generate no more than one output message. The name of the task and latest overlay will be appended to the above message by the REX Message service.

The "ODM" bit may be set by the REX service call invoking overlay loading or through use of the SLO instruction in the loading program. This bit is initially set so that output of messages during initial task loading is done.

The following sections describe each of the instructions which may appear in an LPR in terms of:

1. Function.
2. Format and content.
3. Usage in the load program and effects of usage on the loading process.
4. Error detection.

8.5.1 DEE -- DEFINE (MODULE) ENTRY POINT AND EXIT LOADING PROGRAM

Function:

Defines the entry point of the module, certain task-related status information for task modules loaded as tasks, and the termination point of the loading process.

Format:

Operation Code: #ØØ Instruction Length: 1 or 2 words

The right byte of the first word contains the encoded bit fields as follows with the stated condition indicated when the bit is set to one:

BIT	CODE	USE
8	EPD	Entry point is defined in following word.
9	MOP	Module operates privileged.
1Ø	TIS	Task is symbiont task.

11	CBR	Task can be rolled.
12	TIR	Task is "remote".
14	TIU	Task is "unabortable" (restarts after abort).
15	RRE	Root task must be reloaded on task task exit (for Batch-Processing tasks).

The second word, if EPD is set to one, contains the module entry point.

Usage:

If EPD is set to one and task loading is in progress, the contents of the task's TCBENR word is updated with the contents of the following word added to the load offset. If EPD is set to one, the task's Loader Extension's LOXENO word is updated with the same information. Otherwise, Ø is placed in the applicable field.

If task loading is in progress, the contents of the MOP bit in TCBLFP are used to define the contents of the PRV (Privileged) bit of the task's resource assignment (TCBRES) word. When the task becomes active, the setting of this bit determines whether the task operates in privileged or unprivileged state.

If overlay loading is in progress and control will return to the service invoker after loading, the setting of the MOP bit is not examined.

If overlay loading is in progress and control is to be passed to the module after loading, the transfer of control is accomplished by modifying the REX PS/PR link words as necessary and then executing a CIR instruction. To determine the operational state in which the module will receive control, the MOP bit is examined and...

- If the bit is set to zero, the REX PS link is modified so that the module will operate in the unprivileged state.
- If the bit is set to one and the PRV bit of the TCBRES field is set to one, the REX PS link is modified so that the module will operate in the privileged state.
- If the bit is set to one and the PRV bit of the TCBRES field is set to zero, a load error is generated.

If task loading is in progress, the contents of Bits 1Ø-15 of the first word of the instruction are used to update the contents of the invoking task's TCB. If task loading is not expected, the contents of these bits are ignored.

The ML then inspects the setting of the LEO bit. If the bit is set to zero, no load errors have occurred and control is either returned to the service invoker or passed to the module as specified. If the bit is set to one, the setting of the ML-defined AOE (Abort on Errors) bit is inspected and...

- If AOE is set to one, the REX Abort service is invoked with a reason code of UMF (Unusable Module Found).
- If AOE is set to zero and control is to be returned to the service invoker, the condition code is set to indicate load errors occurred. If control is to be passed to the module, the REX Abort service is invoked as described above.

If the task's System Option word (TCBOPT) has the Hold Loader (HL) bit set to one, the REX Message+Hold Service is invoked with the message:

```
LOADER HOLDING... /ttt/ooo
```

ttt represents the task name and ooo represents the overlay name.

Error Detection:

The following errors are detected and the indicated messages are output if not suppressed:

CODE	MEANING
NME	No module entry point defined. Either task loading is in progress or an overlay is being loaded and control is to pass to the module after loading.
PSC	Privileged state conflict. Loading of an overlay is in progress and the MOP bit is set to one, but the invoking task is not privileged.
NSF	No symbiont found. TIS bit is set to one but no Physical Device Table (PDT) with the task's name can be found.

8.5.2 UTI -- USE TASK I-SPACE MAP IMAGE FOR LOADING

Function:

Causes the task's I-space map image to be used to interpret load program instruction addresses during subsequent operations.

Format:

Operation Code: #Ø1 Instruction Length: 1 word

Right byte contains Ø.

Usage:

This instruction may appear at any point in the loading program. The task I-space Map image will be used for load program instruction processing until another map image is selected.

Error Detection:

None.

8.5.3 UTO -- USE TASK O-SPACE MAP IMAGE FOR LOADING

Function:

Causes the task's O-space map image to be used to interpret load program instruction addresses during subsequent operations.

Format:

Operation Code: #Ø2 Instruction Length: 1 word

Right byte contains Ø.

Usage:

Same as UTI.

Error Detection:

None.

8.5.4 DDL -- DEFINE DIRECTORY OF LOAD PROGRAMS

Function:

Permits a directory of load programs that will be accessed by the REX Read Load Program Directory Service (REX,#32/Call Number #12B) to be included in the load program of a module.

Format:

Operation Code: #Ø3 Instruction Length: 3 to 93 or 121 words

The right byte of the first word contains the number of words to follow in the instruction.

The second word is the address where the subroutine directory will be loaded.

The words following the first word consist of single or doubleword entries in the following format:

BIT	USE
Ø	Set to one if the entry is vacant.
1-6	Ignored.
7	Set to zero if 24 bits are used to describe the FPI of the first LPR of the module associated with this entry. Set to one if 16 bits are used to describe the FPI.
8-15	Contains the high order 8 bits of the FPI if Bit 7 is set to zero. Otherwise, ignored.
16-31	Contains the low order 16 bits of the FPI of the first LPR of the associated load module.

Usage:

The ML treats this instruction as a NOP when it is encountered during module loading and simply bypasses the instruction. However, the REX service which accesses the information contained in the instruction places the following limitations upon the instruction in so far as placement in the load program:

1. The first appearance of this instruction must be as:
   a. The second instruction of a task load module's load program.
   b. The first instruction of an overlay load module's load program.
2. If the instruction is followed by any instruction other than that instruction which calls for the reading of another segment of the load program, it is assumed that there are no more instructions of this type in the load program.

Error Detection:

None.

8.5.5 RMO -- READ MODULE OBJECT RECORDS

Function:

Causes initiation of data-chained read to access MOR's.

Format:

Operation Code: #Ø4 Instruction Length: 4 or more words

The right byte of the first word contains the high order 8 bits of the File Position Index (FPI) of the sector containing the first MOR to be read. The second word contains the low order 16 bits of the applicable FPI.

The following doublewords each contain a Transfer Count (TC) and a Transfer Address (TA).

The TA specifies the virtual address into which the first word read from secondary storage will be read into the selected addressing space of the task. The TC specifies how many words will be read into the memory area starting at the location specified by TA.

The TC is represented in the form required for DMP operations. Hence, all TC's contain a 14-bit representation of the number of words to be transferred when that number is in negative, 2's-complement form.

Bit 1 of all TC words is set to one. Bit Ø of all TC words is set to zero except that Bit Ø of the last TC word is set to one to mark the end of the data chain to the DMP.

Usage:

If no map has been selected at the time this instruction is encountered, the task I-space Map image is used.

The loading offset (bias) address specified by the caller is added to each TA of the data chain prior to initiating a read. Hence, if the load module has been edited to occupy the virtual area into which it will be loaded, a load offset of Ø should be specified.

Error Detection:

None.

8.5.6 RLP -- READ LOAD PROGRAM RECORD

Function:

Causes the Module Loader to read the next LPR.

Format:

Operation Code: #Ø5 Instruction Length: 2 words

The right byte of the first word contains the high order 8 bits of the 24-bit FPI of the next LPR. The second word contains the low order 16 bits of the applicable FPI.

Usage:

When this instruction appears, the ML forces the associated read to be carried out in WAIT mode. At this point, any ML loading operations which require ML execution will cease until the next LPR has been read into memory.

Error Detection:

None.

8.5.7 SSO -- SET TASK'S SYSTEM OPTION WORD

Function:

Causes the System Option Word of the invoking task or another task to be replaced by a new value on a selective basis.

Format:

Operation Code: #Ø6 Instruction Length: 5 words

The second and third words contain Ø if the System Option Word of the invoking task is to be modified or the name of the task whose System Option Word is to be modified.

The fourth word contains a 16-bit mask that is extracted from the System Option Word and the fifth word contains a 16-bit mask that is ORed to the System Option Word.

Usage:

This instruction causes the REX Set System Option service to be invoked with the instruction contents.

Error Detection:

The LEO bit is set to one if the invoking task attempts to modify the System Option Word of another task and the invoking task is not compatible with the other task. If message output is not suppressed, the message SOI (System Option Incompatibility) is generated.

8.5.8 SPO -- SET TASK'S PROGRAM OPTION WORD

Function:

Causes the Program Option Word of the invoking task or another task to be replaced by a new value on a selective basis.

Format:

Operation Code: #Ø7 Instruction Length: 5 Words

The second and third words are set to zero if the Program Option Word of the invoking task is to be modified or contain the name of the task whose Program Option Word is to be modified.

The fourth word contains a 16-bit mask that is extracted from the Program Option Word selected.

The fifth word contains a 16-bit mask that is ORed to the Program Option Word selected.

Usage:

This instruction causes the REX Set Program Option service to be invoked with the instruction contents.

Error Detection:

The LEO bit is set to one if the invoking task attempts to modify the Program Option Word of another task and the invoking task is not compatible with the other task. If message output is not suppressed, the message POI (Program Option Incompatibility) is generated.

8.5.9 SNV -- SET NAMED VARIABLES

Function:

Causes a named variable and an associated value to be added to the Task Variables Extension of the TCB of the invoking task or another task.

Format:

Operation Code: #Ø8 Instruction Length: 5 or more words

The right byte of the first word contains the number of variable names and value pairs following.

The second and third words are set to zero if the Variables Extension of the invoking task is referenced or contain the name of the task whose Variables Extension is referenced.

Succeeding doublewords each contain:

1. In the first word, the name (in CAN code) of the variable to be modified.

2. In the second word, the variable value, where the value #8ØØØ means no value.

Usage:

This instruction causes the REX Set Task Variable service to be invoked as often as necessary with the instruction contents.

This REX service attempts to match the variable name with that of an existing variable in the referenced extension. If the name matches, the value specified replaces the existing value. If no match is found, the name and value are added to the referenced extension if enough room exists in the extension to add the new variable.

Error Detection:

The LEO bit is set to one and, if not suppressed, error messages are output as follows for the indicated conditions:

CODE	MEANING
SVI	Set Variable Incompatibility if the invoking task attempts to reference another task and is incompatible with the referenced task.
SVS	Set Variable Space if the referenced task has no Variables Extension or there is not enough room in the Extension to add the pair.

8.5.10 DTV -- DELETE TASK VARIABLES

Function:

Causes named variables to be deleted from the extension of the invoking task or another task.

Format:

Operation Code: #09 Instruction Length: 4 or more words

The right byte of the first word contains the number of variables to be deleted.

The second and third words contain Ø if the invoking task is referenced or contain the name of the task referenced.

The fourth and following words each contain the name of a variable to be deleted from the referenced task's Variables Extension.

Usage:

Appearance of this instruction causes the REX Delete Task Variables service to be invoked repeatedly as necessary.

Error Detection:

The LEO bit is set to one and, if not suppressed, error messages are output as follows for the indicated conditions:

CODE	MEANING
DVI	Delete Variable Incompatibility if the invoking task attempts to delete variables from the extension of a task with which it is incompatible.
DVF	Delete Variable Failure if the referenced extension does not exist or a named variable cannot be found for deletion.

8.5.11 WAR -- WAIT AND RELOCATE MODULE CONTENTS

Function:

Permits a module, edited to occupy a virtual memory area starting with location Ø, to be loaded at some other load offset with all operand addresses of instructions modified accordingly.

Format:

Operation Code: #ØA Instruction Length: 3 or more words

The right byte of the first word contains the number of words of relocation controls which follow.

The second word contains the displacement from the loading offset specified to which the first relocation control field of the first relocation control word pertains.

The third and following words each contain the relocation controls for 16 words in ascending address sequence on a left-to-right basis. A bit set to one causes the load offset specified to be added to the associated module word. A bit set to zero inhibits addition of the load offset.

Usage:

This instruction must appear in a loading program only when a module has not been edited to occupy a fixed position in the virtual memory area into which it will be loaded, and the module contents reference or require definition of addresses within that area.

When this instruction appears, the ML relinquishes until any reading of MOR's in progress has been completed. This instruction must physically follow the RMO instruction which loads the module contents to which it pertains but need not follow the RMO instruction immediately.

When this instruction is executed, the task map image last specified by the load program is used to fetch and store module words. If the invoking task uses two maps, the applicable map must be current or must have been selected prior to execution of this instruction.

Unless otherwise necessary, module loading time will be minimized by using no instructions of this type in a load program until all RMO instructions have appeared.

Error Detection:

The LEO bit is set to one and, if not suppressed, the error message MRF (Module Relocation Failure) is output if execution causes referencing of a virtual location whose associated Access Rights are anything but '11 (Read/Write Usage).

8.5.12 MAR -- MODIFY ACCESS RIGHTS

Function:

Causes the Access Rights (A/R's) of one or more pages of a task map to be modified as requested.

Format:

Operation Code: #0C Instruction Length: 2 words

The right byte of the first word contains bit fields encoded as follows:

BIT	USE
8-9	Code for A/R's to be set as follows: '11 if read/write usage is desired. '10 if read/execute usage is desired. '01 if read-only usage is desired.
14	Set to one if Load Offset specified is to be added to the virtual page number supplied.
15	Set to one if virtual space referenced is defined by the task's O-space Map.

The second word contains:

0-7	Number of the virtual page in the map at which modification is to begin.
8-15	Low order 8 bits of the 2's-complement, negative number of pages in the area (special format, with sign bit missing and understood to be equal to one).

Usage:

When a module is being loaded, the virtual area into which it is being loaded must have been set to A/R's of '11 (read/write). If these A/R's are desired after loading, this instruction need not appear in a load program. Access Rights of '00 (no access) may not be set.

Regardless of the length of the map of the invoking task, a starting page/number of pages specification of "0/256" (represented as #0000) causes all pages of the map of the invoking task to be modified.

The appearance of this instruction causes the ML to invoke the REX Modify Access Rights service with the contents of the instruction.

This instruction may be used to modify the A/R's of pages other than those into which the module is being loaded. Shared pages may not have their A/R's modified in this manner.

Error Detection:

The LEO bit is set to one and, if not suppressed, an error message of ARV (Access Rights Violation) is output if the REX service refuses to modify the access rights.

8.5.13 IGS -- INSERT GLOBALLY DEFINED SHARED PAGE REGION

Function:

Causes the actual page(s) specified by a definition entered in the system's Global Shared Page Directory to be edited into a map of the invoking task in a specified area of virtual memory.

Format:

Operation Code: #0D Instruction Length: 7 words

The right byte of the first word contains the encoded bit fields:

BIT	USE
8-9	Code for A/R's to be set as follows: '11 if read/write usage is desired. '10 if read/execute usage is desired. '01 if read-only usage is desired.
14	Set to one if the upper 8 bits of the Load Offset specified is to be added to the starting page number specified in the instruction.
15	Set to zero if the task's I-space Map Image is to be edited or set to one if task's O-space Map Image is to be edited.

The second word contains the CAN code representation of the Read Key for the region or zero if none is to be specified.

The third word contains the CAN code representation of the Write Key for the region or zero if none is to be specified.

The fourth, fifth, and sixth words contain the first 6 ASCII bytes of the name of the shared region referenced.

The seventh word contains the virtual address at which the region definition is to be inserted (low order 8 bits must be zero) in the selected task addressing space.

Usage:

The appearance of this instruction causes the REX Insert Global Shared Region service to be invoked. The service is invoked with the contents of the instruction after the Load Offset specified has, if necessary, been added to the virtual address supplied.

If this instruction appears in a loading program, the invoking task's TCB must have a Page Sharing Extension at the time the instruction appears. An entry in this extension must be "filled-out" so that the system will be able to dispose of the shared region should the task exit or abort.

Error Detection:

The LEO bit is set to one and, if not suppressed, the error message IGF (Insert Global Failure) is output if the REX service refuses to or cannot insert the area.

8.5.14 EGS -- Extract Global Shared Region(s)

Function:

Causes one named Global Shared Region or all Global Shared Regions existing in a virtual region of a task map to be removed from the virtual space and made inaccessible to the task.

Format:

Operation code: #0E Instruction Length: 4 words

The right byte of the first word contains the encoded bit fields:

BIT	USE
14	Set to one if the Load Offset should be added to the virtual region beginning address.
15	Set to one if the virtual region specified exists in the task's O-space Map.

The second, third, and fourth words contain either one or the other of the following:

- Bytes 1 through 6 of the name of the specific region to be extracted.
- A virtual region description specified as Starting Page Number/Number of Pages in the fourth word with the uppermost byte containing the page number where the region starts and the lower byte containing the negative number of pages (in a special format with sign missing and understood to be equal to one). The second and third words contain zero.

Usage:

This instruction causes shared pages to be extracted from a map image and the corresponding virtual region to be made available for reuse.

The appearance of this instruction causes the REX Extract Shared Region(s) service to be invoked with the contents of the instruction.

Regardless of the size of a map, a virtual range specified as "Ø/256" (represented as #ØØØØ) will cause all Global Shared Regions to be extracted from a map.

Error Detection:

The LEO bit is set to one and, unless suppressed, the error message EGF (Extract Global Failure) is generated if the REX service returns any condition code except that meaning "operation was completed successfully".

8.5.15 CPR -- CREATE PRIVATE SHARED REGION

Function:

Causes the definition of a Private Shared Region to be created and entered in the system's Private Shared Region Directory.

Format:

Operation Code: #ØF Instruction Length: 7 words

The right byte of the first word contains bits fields encoded as follows:

BIT	USE
8-9	Code for A/R's to be set as follows: '11 if read/write usage is desired. '1Ø if read/execute usage is desired. 'Ø1 if read-only usage is desired.
1Ø	Set to one if no task will be allowed to access the region for read/write usage.
14	Set to one if the Load Offset specified should be added to the region starting page number.
15	Set to zero if the region exists in the task's I-space Map Image or one if in the task's O-space Map Image.

The second word contains the CAN code representation of the Read Lock to be applied to the region or zero if no lock is to be defined.

The third word contains the CAN code representation of the Write Lock to be applied to the region or zero if no lock is to be defined.

The fourth, fifth, and sixth words contain, respectively, bytes 1 through 6 of the Private Shared Region name.

The seventh word defines the virtual region in the task's memory.

BIT	USE
Ø-7	Contains starting virtual page number (Ø-255).
8-15	Contains the number of pages in the region (1-256) in negative, 2's-complement representation, where the sign bit is missing and understood to be equal to one.

Usage:

The appearance of this instruction causes the REX Create Private Shared Region service to be invoked with the contents of the instruction after, if necessary, the Load Offset specified is added to the starting virtual page.

Error Detection:

The LEO bit is set to one and, unless suppressed, the error message CPF (Create Private Shared Region Failure) is output if the REX service refuses to or cannot perform the requested service.

8.5.16 IPS -- INSERT PRIVATE SHARED REGION

Function:

Makes a Private Shared Region accessible to a task in a specified region of the task's virtual memory.

Format:

Operation Code: $10 Instruction Length: 7 words

The right byte of the first word contains bit fields encoded as follows:

BIT	USE
8-9	Contain codes for Access Rights requested:
	'11 if read/write access requested.

'1Ø if read/execute access requested.

'Ø1 if read-only access is requested.

14 Set to one if the Load Offset specified is to be added to the virtual region starting page number specified.

15 Set to zero if specified region exists in the task Instruction Map space or set to one if the region exists in the task O-space Map space.

The second word contains the CAN code representation of the Read Key to be applied to the Read Lock on the region or Ø if no key value is supplied.

The third word contains the CAN code representation of the Write Key to be applied to the Write Lock on the region or Ø if no key value is supplied.

The fourth, fifth and sixth words contain, respectively, bytes 1 through 6 of the Private Shared Region ASCII name.

The seventh word contains the address at which insertion of the region in the task's virtual memory is to begin. The low-order 8 bits of the address must be zero.

Usage:

The appearance of this instruction causes the REX Insert Private Shared Region service to be invoked with the contents of the instruction after, if requested, the Load Offset is added to the virtual address specified in the instruction.

Error Detection:

The LEO bit is set to one and, if not suppressed, the error message IPF (Insert Private Shared Region Failure) is output if the REX service refuses to or cannot perform the service requested.

8.5.17 EPS -- EXTRACT PRIVATE SHARED REGION(S)

Function:

Causes a single Private Shared Region or all Private Shared Regions in a specified region of virtual memory to be made inaccessible and all associated regions to be made free regions.

Format:

Operation Code: #11 Instruction Length: 4 words

The right byte of the first word contains bit fields encoded as follows.

BIT	USE
14	Set to one if a virtual region is specified and the Load Offset is to be added to the starting region address specified.
15	Set to zero if a virtual region is specified and exists in memory defined by the task's I-space Map or set to one if in memory defined by the task's O-space Map.

If a single Private Shared Region is to be extracted, the second, third, and fourth words contain bytes 1 through 6 of the name of the region.

If any (all) Private Shared Region(s) in a specified virtual region are to be extracted, the second and third words are set to zero and the fourth word defines the starting page and number of pages in the region (in the special notation "starting-page/negative-number-of-pages").

Usage:

The appearance of this instruction causes the REX Extract Shared Region(s) service to be invoked with the contents of the instruction after, if necessary, the Load Offset is added to the starting page number of the region specified in the instruction.

Error Detection:

The LEO bit is set to one and, unless suppressed, the error message EPF (Extract Private Shared Region Failure) is output if the REX service provides any condition code upon return indicating anything but "operation completed successfully".

8.5.18 ISM -- INSERT SHARED LOAD MODULE

Function:

Makes a Shared Load Module accessible in a specified region of the task's virtual memory.

Format:

Operation Code: #12 Instruction Length: 7 words

The right byte of the first word contains bit fields encoded as follows:

BIT	USE
8-9	Code for access rights requested for region: '11 if read/write access is requested. '10 if read/execute access is requested. '01 if read-only access is requested.
10	Set to one if no task will be allowed to access the region for read/write usage.
14	Set to one if the Load Offset specified is to be added to the starting page number of the specified region.
15	Set to zero if region exists in virtual memory defined by task's I-space Map Image or set to one if region exists in memory defined by task's O-space Map Image.

Depending on whether the module of the shared region must be loaded or if the region is already defined, the contents of the second and third words are considered as Read and Write Locks or Keys.

The fourth word contains the LFN of the file in which the module is contained.

The fifth and sixth words contain CAN code characters 1 through 6 of the name of the load module.

The seventh word specifies the address in the virtual region into which the module is to be loaded or in which the module is to be made accessible. The low order byte of this word must be zero.

Usage:

The appearance of this instruction causes the REX Insert Shared Load Module service to be invoked with the contents of the instruction. This may invoke the ML recursively.

Error message output depends on the setting of the ODM bits during the current loading process. The REX service is not allowed to abort if a load error occurs but, in event of a load error, the LEO bit is set to one.

Either the module called must be contained in a Global file or a file definition with the referenced LFN must appear as part of the TCB of the task when this instruction is encountered.

Error Detection:

The LEO bit is set to one and, if not suppressed, the error message IMF (Insert Shared Module Failure) is output if the REX service refuses to or cannot perform the requested service.

8.5.19 ESM -- EXTRACT SHARED LOAD MODULE(S)

Function:

Causes a single Shared Load Module or any Shared Load Modules in a specified virtual region to be made inaccessible and all associated regions to be made free regions.

Format:

Operation Code: #13 Instruction Length: 3 words

The right byte of the first word contains bit fields encoded as follows:

BIT	USE
14	Set to one if a virtual region is specified and the Load Offset is to be added to the number of the starting page of the region.
15	If a virtual region is specified, the bit is set to zero if the region exists in memory defined by task's I-space Map or set to one if the region exists in memory defined by task's O-space Map.

If a single module is to be extracted, the second and third words contain CAN code characters 1 through 6 of the Shared Load Module ASCII name.

If any shared modules are to be extracted from a region of virtual memory, the second word contains zero and the third word contains the starting virtual page number and number of pages in the region.

Usage:

The appearance of this instruction causes the REX Extract Shared Region(s) service to be invoked with the contents of the instruction after, if necessary, the Load Offset is added to the starting page number of the region.

Error Detection:

The LEO bit is set to one and, if not suppressed, the error message EMF (Extract Shared Module Failure) is output if the REX service provides a condition code indicating anything but "operation completed successfully".

8.5.20 ERA -- ERASE ALL SHARED PAGE REGIONS

Function:

Causes any shared page regions in a specified virtual region to be made inaccessible and any (all) associated regions to be made free regions.

Format:

Operation Code: #14 Instruction Length: 2 words

The right byte of the first word contains bit fields encoded as follows:

BIT	USE
14	Set to one if Load Offset is to be added to the starting page number of the region.
15	Set to zero if the region exists in memory defined by the task's I-space Map or set to one if the region exists in memory defined by the task's O-space Map. If the region description is zero, describing an entire map, extended memory regions will also be erased.

Usage:

The appearance of this instruction causes the REX Extract Shared Region(s) service to be invoked with the contents of the instruction after the Load Offset is added to the starting page number of the region.

Error Detection:

The LEO bit is set to one and, if not suppressed, the error message ERF (Erase Failure) is output if the REX service returns a condition code indicating anything but "operation completed successfully".

8.5.21 ALO -- ALLOCATE PRIVATE PAGES

Function:

Causes a region of task virtual memory to be modified such that:

1. Private pages are allocated to all non-shared virtual pages of the region.

2. All private pages have Access Rights of '11 (Read/Write Usage Permitted).

Format:

Operation Code: #15 Instruction Length: 2 words

The right byte of the first word contains encoded bit fields as follows:

BIT	USE
9	Used only by TOC for user-specified allocation.
13	Set to zero if virtual region describes memory into which MOR contents will be loaded.
14	Set to one if the Load Offset is to be added to the starting page number specified for the area.
15	Set to one if the space referenced exists in the task's 0-space Map.

The second word contains:

BIT	USE
0-7	Starting page number of the region.
8-15	Number of pages in the region specified as the low-order 8 bits of the 2's-complement, negative representation of the number of pages in the area.

Usage:

Regardless of the size of the virtual space defined by the referenced map, a Starting Page/Number of Pages specification of "0/256" (represented as #0000) causes all pages of the region, as defined by the map image of the region, to be modified as necessary.

This instruction is always generated by the MAX IV Task/Overlay Cataloger for each virtual region into which MOR's will be loaded. Bit 13 is set to have the region examined to ensure that:

1. The region described does not lie in the MAX IV Operating System Nucleus region of MAP Ø.

2. Any virtual pages of the region that are shared have A/R's that indicate "read/write" usage is permitted. This permits the ML to take action to avoid having the loading process aborted by the MAX IV Basic I/O System because an attempt is made to read MOR contents into virtual regions having A/R's indicating "read/write" usage is not permitted.

If Bit 13 is set and a violation of the above conditions is found, a load error is generated and the REX Allocate service is not invoked. Otherwise, the service is invoked using the contents of the instruction.

Allocation for regions other than those into which MOR contents will be loaded may be invoked by use of this instruction with or without region checking triggered by Bit 13.

Error Detection:

If a load error is generated as a result of checks initiated by the contents of Bit 13, the LEO bit is set to one and, unless suppressed, one of the following error messages is output:

CODE	MEANING
ANO	Attempted Nucleus Overlay.
RNL	Region Not Loadable.

If the REX Service is invoked and refuses to allocate the pages because the region specified does not lie wholly within the virtual space defined by the referenced map, the LEO bit will be set to one and, unless suppressed, the error message ALF (Allocate Failure) is output.

If the LEO bit is set to one under any circumstances, the ML internal signal bit MAF (Memory Allocation Failed) is also set to one. Once this bit is set to one, any instructions of the following types that are subsequently encountered in the loading process are bypassed by the ML:

1. All RMO, IMR, WAR, and SAC instructions.

2. SSO, SPO, SNV, and DTV instructions that reference other tasks.

8.5.22 DEA -- DEALLOCATE PRIVATE PAGES

Function:

Causes all private, non-shared actual pages associated with a specified virtual region to be deallocated.

Format:

Operation Code: #16 Instruction Length: 2 words

Refer to the ALO instruction for the rest of the format.

Usage:

Regardless of the size of the virtual space defined by the referenced map, a Starting Page/Number of Pages specification of "0/256" (represented as #0000) causes all private pages of a map to be deallocated.

The appearance of this instruction causes the REX Deallocate service to be invoked with the contents of the instruction.

Error Detection:

The LEO bit is set to one and, unless suppressed, the error message DEF (Deallocation Failure) is output if the REX service returns any condition code other than that indicating "operation completed successfully".

8.5.23 DMR -- DEFINE MODULS'S VIRTUAL RANGES

Function:

Provides information required to update the Loader Space Usage Extension of the task's TCB. The information relates to the virtual space defined by either the task's I-space Map or O-space Map and consists of:

1. Lowest virtual address into which module contents are loaded.
2. Highest virtual address into which module contents are loaded.
3. Highest virtual address currently accessible to the task in the map after module loading is completed.

Format:

Operation Code: #17 Instruction Length: 4 words

The right byte of the first word contains zero if information pertains to the task's I-space Map or 1 if information pertains to the task's O-space Map.

The second, third, and fourth words contain, respectively, the information defined above.

Usage:

The Load Offset is added to all addresses supplied in the instruction. The low and high addresses unconditionally replace the contents of the appropriate fields of the Loader Extension.

The highest currently accessible address is inserted into the Loader Space Usage Extension if not inhibited by a one contained in Bit 15 of the LOXMAI or LOXMAO field of the extension.

Error Detection:

None.

8.5.24 FAR -- REASSIGN A FAT ENTRY

Function:

Causes the Current or Default assignment or both assignments of an existing FAT entry to be modified.

Format:

Operation Code: #19 Instruction Length: 4 words

Bit 15 of the first word is set to one if the FAT is to be classified as a "temporary" FAT.

The second word contains the LFN of the file for which the assignment(s) is to be modified.

The third word contains the name of the logical device to which the current assignment of the file is to be changed or zero if the current assignment is not to be changed.

The fourth word contains the name of the logical device to which the default assignment of the file is to be changed or zero if the default assignment is not to be changed.

Usage:

The contents of the instruction are used to selectively invoke the REX Assign Service (REX,#32/Call Number #A).

The execution of this instruction may affect subsequent loading operations but cannot affect those operations initiated by instructions appearing in the load program prior to this instruction.

Error Detection:

The LEO bit is set to one and, unless suppressed, the error message FAR (FAT Redefinition Error) is output if the REX Assign service refuses to or cannot perform the requested service.

8.5.25 NOP -- NO OPERATION

Function:

Permits system elements which generate load modules to insert information in the LPR's for their own use in a fashion that is transparent to the ML.

Format:

Operation Code: #1B Instruction Length: Variable

The right byte of the first word contains the number of words following.

The balance of the format is determined by the generating program.

Usage:

The usage is determined by the generating system element. The ML simply bypasses the following words during the loading process.

Error Detection:

None.

8.5.26 SAC -- SET AREA CONSTANT

Function:

To initialize an area of virtual memory to a constant value.

Format:

Operation Code: #1D Instruction Length: 4 words

The right byte of the first word contains zero if the virtual address supplied references space defined by the task's I-space Map or contains one if the address references space defined by the task's O-space Map.

The second and third words contain, respectively, the virtual addresses of the first and last words constituting the area to be initialized.

The fourth word contains the value to which the area is to be initialized.

Usage:

The Load Offset specified at loading initiation will be added to both addresses supplied.

Error Detection:

The LEO bit is set to one and, unless suppressed, the error message SCF (Set Constant Failure) is output if execution of this instruction causes a virtual location to be referenced that has A/R's of anything but '11 (Read/Write Usage Permitted).

8.5.27 ICM -- INSERT COPYABLE LOAD MODULE

Function:

Causes another load module to be loaded.

Format:

Operation Code: #1E Instruction Length: 5 words

The right byte of the first word contains bit fields encoded as follows:

BIT	USES
8	Set to one if the loader extension is not to be updated by this call to the loader. Set to zero if the loader extension is to reflect the module loaded.
14	Set to one if the load offset specified is to be added to the starting page number of the load address.

The second word contains the address at which the module is to be loaded.

The third and fourth words contain CAN code characters 1 through 6 of the name of the load module.

The fifth word contains the LFN of the file from which the module is to be loaded.

Usage:

The appearance of this instruction causes the REX Load Named Overlay service to be invoked with the contents of the instruction after the address is added to the address specified in this instruction.

Refer to the ISM instruction for the remainder of the usage.

Error Detection:

Same as ISM except, if not suppressed, the error message ICM (Insert Copyable Module Failure) is output.

8.5.28 CTA -- CLEAR TEMPORARY FAT ASSIGNMENTS

Function:

Causes all FAT entries of the task that have been designated as "temporaries" to be made available for reuse.

Format:

Operation Code: #1F Instruction Length: 1 word

The right byte of the instruction is ignored.

Usage:

The appearance of this instruction causes any FAT entry of the task that was created as a "temporary" to be made vacant and, hence, to be available for reuse by the task.

Error Detection: None.

8.5.29 SOB - START/STOP OBJECT AND CHANGE MAP MODES

Function:

Delimits information by class for the MAX IV Task/Overlay Cataloger. Also, informs the loader in which map mode the module is to run.

Format:

Operation Code: #21 Instruction Length: 1 word

The right byte of this instruction contains bit fields encoded as follows:

BIT	USE
8	Set to one if module is to run in the 2-map mode.
9	Set to one if module is to run in the CLASSIC 78xy Pipeline mode.
1Ø	Set to one if module is relocatable.
11	Set to one if resume on I/O complete.
13	Set to one if first SOB. This bit is used by TOC but ignored by the module loader.
14	Set to one if code is generated by M4EDIT.
15	Set to one if this is the first appearance of this instruction.

Usage:

If Bit 15 of the first word is zero, this instruction is treated as a NOP by the Module Loader.

If Bit 15 is one, this instruction causes the Module Loader to make the loaded module:

- Run in the 2-map mode if Bit 8 equals 1.
- Run in the 1-map mode if Bit 8 equals Ø.

Error Detection:

If the prologue TCB of the task was not generated for 2-map mode execution and 2-map mode module execution is requested, then, if not suppressed, the error message NTM (Not a 2-Map Task) is output.

8.5.3Ø SPD -- SET PROLOGUE DEFINITIONS

Function:

Defines information needed to allocate resources to a task whose body is being loaded and constructs as much of the TCB as is needed to initiate the loading process.

Format:

Operation Code: #3Ø Instruction Length: 12 words

The right byte of the first word contains bit fields encoded as follows:

BIT	USE
9	Set to one if the task has been cataloged for MAX IV Revision F or later.
1Ø	Set to one if the task is privileged.
11	Set to one if the task is a Symbiont task.
12	Set to one if the task is to Auto Start or zero if the task is activated by some event other than appearance of its Prologue TCB in the CPU queue.
13	Set to one if the task does take exclusive use of a register block and does not need a Register Save Area Extension of its TCB.
14	Set to one if only an influence level is cataloged for the task.
15	Set to one if no execution priority is cataloged for the task.

The following words contain:

WORD	CONTENTS
2	Left byte contains cataloged influence level. Right byte contains cataloged execution priority.
3	Bit 1 is set to one if the task uses only one map or set to zero if the task uses two maps. Bits 5-7 contain the number of pages in excess of 1 page that must be allocated for the task's EXT area. All other bits are set to zero.
4	Number of pages used by the task in its I-space Map represented as a negative, 2's-complement number.
5	Number of pages used by the task in its O-space Map represented as a negative, 2's-complement number.
6	Number of words to be allocated for the task's Main Push Stack. Bit Ø is set to one if the task's main push stack is not extensible. Bits 1 through 7 are reserved.

7 Number of words to be allocated for the task's PS/PR Push Stack.

Bits Ø through 7 are reserved.

8 The left byte (bits Ø-7) contains the maximum number of Global I/O nodes the task can use. The right byte (bits 8-15) contains the number of private I/O nodes over one that the Task allocates when the task is loaded.

9 Reserved.

1Ø Reserved.

11 Reserved.

12 Reserved.

Usage:

This instruction must be the first instruction of the first LPR of a task's load module.

For nonresident tasks, the presence of this instruction is determined when the task is scheduled. The MAX IV Operating System element named PS$TSU actually executes this instruction in the task start-up process prior to actual loading of the task by the ML. Such execution is by the scheduling task and not the scheduled task.

When loading, the ML treats this instruction as a NOP and bypasses the balance of the instruction.

Error Detection:

None by the ML.

8.5.31 DPS -- DEFINE PAGE SHARING EXTENSION

Function:

Causes an area of the task's TCB area to be allocated and initialized for use as the Page Sharing Extension.

Format:

Operation Code: #32 Instruction Length: 1 word

The right byte contains the number of entries to be reserved.

Usage:

The required number of words are allocated and the address of the area is placed in the invoking task's TCBPSI word.

The first word of the area is initialized to the address of the last word of the area. The PSESTA field of each entry of the area is initialized to indicate the entry is not in use.

Error Detection:

None.

8.5.32 DCA -- DEFINE COMPATIBLE ALLOCATION EXTENSION

Function:

Causes an area of the task's TCB area to be allocated and initialized for use as the Compatible Allocation Extension.

Format:

Operation Code: #33 Instruction Length: 1 word

The right byte contains the number of words to be reserved.

Usage:

The required number of words are allocated and the address of the area is placed in the invoking task's TCBCAL word.

The first word of the area is initialized to the address of the third word of the area. The second word of the area is initialized to the address of the first word of the area plus the length of the area for use by the REX Compatible Allocate/Deallocate services.

Error Detection:

None.

8.5.33 DVE -- DEFINE VARIABLES EXTENSION

Function:

Causes an area of the task's TCB area to be allocated and initialized for use as the Variables Extension and causes the name and value of a standard variable to be inserted in the area.

Format:

Operation Code: #34 Instruction Length: 1 word

The right byte contains the number of words to be reserved.

Usage:

The required number of words is allocated and the address of the area is placed in the invoking task's EXTVAR word.

Word 1 of the area is initialized to the value 1. Word 2 of the area is initialized to a value equal to the number of words reserved less 2 words, the remainder being divided by 2.

Word 3 of the area is initialized to zero which is equivalent to having a variable named @bbb. Word 4 of the area is initialized to the value #8000. This first variable is known as the "unspecified" variable since its name is allowed to change continuously in use. All other variables are known as "specified" variables since their names are fixed.

Error Detection:

None.

8.5.34 DUC -- DEFINE TASK UTILIZATION LIMIT/COUNT

Function:

Causes the value of the invoking task's TCBUTM field to be initialized.

Format:

Operation Code: #36 Instruction Length: 3 words

The right byte of the first word of this instruction contains bit fields encoded as follows:

BIT	USE
15	Set to one if the task is to abort instead of continuing when Task Utilization Limit is reached.

The second and third words of the instruction contains the 32-bit value that is to be placed in the referenced TCB field when the task is initially loaded.

Usage:

The contents of the TCBUTM word will not be modified if a task's body is being reloaded.

Error Detection:

None.

8.5.35 FAV -- DEFINE 1-N VACANT FAT ENTRIES

Function:

Causes the creation of one or more vacant, reuseable FAT entries in the task's TCB area.

Format:

Operation Code: #37 Instruction Length: 1 word

The right byte of the instruction word contains the number of FAT entries to be created.

Usage:

For each FAT created:

1. Space for the entry is assigned in the task's TCB area.

2. The address of the entry is entered into the task's Logical File Index Table.

3. The entry area is cleared to contain all zeros.

Each entry created by this instruction is marked as a temporary and will be cleared for reuse if a CTA instruction is encountered in any load module used by the task.

Error Detection:

None.

8.5.36 FAD - DEFINE THE ASSIGNMENTS OF ONE FAT

Function:

Causes the creation of one FAT and the initialization of the assignments of the FAT.

Format:

Operation Code: #39 Instruction Length: 4 words

The right byte of the first word contains 1 if the FAT is to be defined as a temporary FAT and Ø if the FAT is to be defined as permanent.

The second word contains the Logical File Name to be associated with the FAT.

The third word contains the name of the device or logical file that is to be used for the current assignment of the logical file associated with the FAT or Ø if no current assignment is to be made. The fourth word contains the name of the device

or logical file that is to be used for the default assignment of the logical file associated with the FAT or Ø if no default assignment is to be made.

Usage:

The appearance of this instruction causes:

1. The area for a FAT entry to be assigned in the task's TCB area.

2. The address of the FAT to be entered in the Logical File Index Table of the task.

3. The area to be cleared to Ø.

4. The REX Assign service to be invoked to make the current and/or default assignments indicated by the contents of words 3 and 4 of the instruction.

If the FAT is not to be defined as temporary, the permanent status bit of the FAT status word is set to one.

Error Detection:

The LEO bit is set to one and, unless suppressed, the error message FAD (FAT Definition Error) is output if the REX Assign service indicates an assignment error has occurred.

8.5.37 DFL -- DEFINE THE LOGICAL FILE INDEX TABLE (LIST) EXTENSION

Function:

Causes space for the Logical File Index Table extension of the task's TCB to be allocated and initialized for use.

Format:

Operation Code: #3B Instruction Length: 1 word

The right byte of the instruction word contains the number of words to be reserved for the extension (the maximum number of entries required plus 2).

Usage:

The appearance of this instruction causes:

1. The number of words necessary to be allocated in the task's TCB area.

2. The address of the area to be inserted in the TCBFAT word of the task's TCB.

3. The entire assigned area to be cleared to Ø.

4. The maximum number of entries permitted to be inserted as the first word of the assigned area.

Error Detection:

None.

8.5.38 SPL -- DEFINE SYSTEM VIRTUAL PAGE LIST EXTENSION

Function:

Allocates and initializes an area of TCB space to be used for the System Virtual Page List extension of the task's TCB.

Format:

Operation Code: #3C Instruction Length: 1 word

The right byte of the instruction word contains the number of words to be reserved for the extension (maximum number of entries the extension is to hold plus 1).

Usage:

The assigned area is initialized to contain:

1. Maximum number of entries permitted in the first word.

2. Zero in all other words.

The address of the extension is placed in the TCBSPL field of the task's TCB.

Error Detection:

None.

8.5.39 UDC -- GO TO USER-DEFINED LOADER CODE

Function:

Call a user's program as an extension to the loader.

Format:

Operation Code: #22 Instruction Length: Variable

The right byte of the first word of this instruction contains the number of words following the first word that are part of the instruction.

Usage:

The usage is dependent upon the user-defined loader extension:

U.LDR User Defined Loader Routine

This routine is called as an extension to the MAX IV Module Loader.

On entry the register contents are:

REGISTER	CONTENTS
1	TCB address of calling task.
2	Address of XUFT in stack.
3	-
4	Op Code of load command in bits 8-15.
5	-
6	Number of words following loader command word.
7	Word immediately following loader command word.
8	Link with initial caller of loader.
9	Address of 2nd UFT.
10	Loader control flags.
11	Address of 1st word after loader command word.
12	-
13	-
14	-
15	-

On exit, R1, R2, R8, R9, and R10 must be unmodified. R11 must be advanced to the next loader command. R3, R4, R5, R6, R7, R12, R13, R14, R15 may be modified.

The exit should be to either ML$NXT (to get next command) or ML$ERR (to process a loader error). If the exit is to ML$ERR, R4 must contain the CAN code error reason code.

Error Detection:

May be supplied in user code.

8.5.40 USR -- DEFINE A USER EXTENSION IN THE EXT OF A TASK

Function:

Add an extension to the EXT of a task that is reserved for the user.

Format:

Operation Code: #3D Instruction Length: 1 word

The right byte of the instruction word contains the number of words to be reserved in the extension (giving a maximum extension size of 255 words). Word Ø of the extension is set to the number of words in the extension by the Module Loader.

Usage:

The appearance of this instruction causes:

1. The number of words needed to be allocated in the task's EXT.
2. The extension to be cleared to zero.
3. The first word of the extension to be set to the extension size.

Error Detection:

None.

8.5.41 WKB -- Define Workbench Extension Wanted

Function:

Add the workbench extension for this task.

Format:

Operation Code: #3E Instruction Length: 1 word

Usage:

The appearance of this instruction causes:

1. Twelve words of the extension to be set aside for the workbench extension.
2. The extension to be cleared to zero.
3. Word TCBKIN of the task control block to point to the extension.

Error detection:

None.

8.5.42 CLE -- DEFINE COMMAND LINE EXTENSION

Function:

Add the Command Line Extension for this task.

Format:

Operation Code: #3F Instruction Length: 1 word

Usage:

The appearance of this instruction causes:

1. Forty-one words of the extension to be set aside for the command line extension.

2. The extension to be cleared to zero except for first word which contains 41.

3. Word EXTCLE of the Task Control Block extension to point to the Command Line Extension.

Error Detection:

None.

8.6 MODULE LOADER ERROR MESSAGE CODES

CODE	MEANING	INSTRUCTION
ALF	Allocate Failure	ALO
ANO	Attempted Nucleus Overlay	ALO
ARV	Access Rights Modification Failure	MAR
CPF	Create Private Shared Region Failure	CPR
DEF	Deallocation Failure	DEA
DVF	Delete Variable Failure	DTV
DVI	Delete Variable Incompatibility	DTV
EGF	Extract Global Failure	EGS
EMF	Extract Shared Module Failure	ESM
EPF	Extract Private Shared Region Failure	EPS
ERF	Erase Shared Region Failure	ERA
FAD	FAT Definition Error	FAD
FAE	File Assignment Error. The next message output defines the error generated as follows. LFN=xxx NOW=yyy DEF=zzz where LFN is the file being assigned, NOW is the now assignment being made and DEF is the default assignment being made. Either yyy or zzz is in error.	
FAR	FAT Redefinition Error	FAR
ICM	Insert Copyable Module Failure	ICM
IGF	Insert Global Failure	IGS
IMF	Insert Shared Module Failure	ISM
IPF	Insert Private Shared Region Failure	IPS
MAF	Memory Allocation Failed	ALO
MRF	Module Relocation Failure	WAR
NME	No Module Entry Point	DEE
NSF	No Symbiont PDT Found	DEE
NTM	Not a 2-Map Task	SOB

POI	Program Option Incompatibility	SPO
PSC	Privileged State Conflict	DEE
RNL	Region Not Loadable	ALO
SCF	Set Constant Failure	SAC
SOI	System Option Incompatibility	SSO
SVI	Set Variable Incompatibility	SNV
SVS	Set Variable Space	SNV

8.7 MODULE LOADER TRACE

When debugging modified versions of the Module Loader or the Task Overlay Cataloger (TOC), errors can occur during task or overlay loading which might cause a system halt. The normal response to this would be to take a system dump and analyze it to see how far along the load was. Unfortunately, if the first error caused other errors, this cascading effect can make the dump almost useless.

The Module Loader Trace provides the service to trace each loader instruction before it is processed. It can also be made to hold prior to processing the loader instruction at the point where the trace occurs. With these features, the loading of the module can be "walked" until the error occurs, thereby, isolating the loader function at fault.

The word at the internal definition ML$TRF controls the Module Loader Trace function when used in conjunction with the task Hold-on-Load (HL) option. If the HL option is on and Bit 15 at location ML$TRF is on, the Module Loader will trace. If Bit 14 is also on, a hold occurs each time a trace output occurs. Tracing can be disabled by resetting the task's HL option.

The output of the trace is as follows:

```
ML TRACE: IC=#ii OP=#oo LPR FPI=ddddd NAM=xxxxxx
```

Where:

ii	is the instruction sequence number from the TOC map.
oo	is the loader operation code from the TOC map.
ddddd	is the File Position Index of the load program record currently being processed.
xxxxxx	is the name of the module being processed.

8.8 SEQUENTIAL LOADER OPTION

The Sequential Task Loader is an optional alternative to the Module Loader. It provides a limited subset of the task activation capabilities of the current Module Loader, while adding the capability to activate tasks stored on sequential media. This capability is added for users who are running process control systems in harsh environments, with the intention of permitting them to run Max IV Systems without discs.

To include the Sequential Task Loader rather than the Module Loader in the system, the MLSEQUENTIAL statement must be included in the system block of the SYSGEN. This statement has one parameter that must be the global logical file from which the tasks are to be loaded. A system that is to use the Sequential Loader must have no preschedule statements, and must not include resident directories.

The Sequential Task Loader expects to find one or more TOC GET's with a single file mark to terminate on the logical file specified on the MLSEQUENTIAL statement. The Sequential Task Loader will be invoked by the X task to load each of these tasks.

APPENDIX A
FEATURES IN THE MAX IV OPERATING SYSTEM

This appendix contains a chart that provides an overview of the MAX IV Operating System. The legend at the bottom of the chart explains the symbols used.

FEATURE	MAX IV
RESIDENT REAL-TIME FOREGROUND TASKS....	X
NONRESIDENT OVERLAY PROGRAMS.....	X
TRANSIENT NONRESIDENT TASKS.....	X
SCHEDULES CPU TIME ON PRIORITY BASIS...	X
SCHEDULES CPU TIME ON ROUND-ROBIN BASIS	X
I/O OPERATIONS CENTRALLY SCHEDULED.....	X
I/O DEVICES ACCESSED LOGICALLY.....	X
DYNAMIC MEMORY ALLOCATION.....	X
STANDARD BATCH PROCESSING TASK(S).....	X
MULTIPLE MEMORY PARTITIONS AND POOLS...	N/A
AUTOMATIC ROLL-OUT LOW PRIORITY TASKS..	X
256 UNIQUE TASK PRIORITY LEVELS	X
SPECIAL PROGRAMS CAN EXECUTE "ABOVE" OPERATING SYSTEM.....	X
RESIDENT OPERATOR DIRECTIVES.....	X
NONRESIDENT OPERATOR DIRECTIVES.....	X
REENTRANT EXECUTIVE SERVICES.....	X
NONRESIDENT EXECUTIVE SERVICES.....	o
I/O HANDLERS FOR ALL STANDARD DEVICES..	X
HANDLERS FOR PROCESS I/O DEVICES.....	X
HANDLERS FOR COMMUNICATIONS DEVICES....	X
I/O QUEUED ACCORDING TO TASK PRIORITY..	X
SYSTEM GENERATION PERFORMABLE ONLINE...	X
SYMBIONT TASKS SUPPORTED.....	X
OUTPUT SPOOLING FOR PRINTERS(OPTIONAL).	X
INPUT SPOOLING OF BATCHED CARDS (OPT.).	X
ABSOLUTE GLOBAL COMMON REGIONS.....	X
MAPPABLE GLOBAL COMMON REGIONS.....	X
PRIVATE INTER-TASK COMMON REGIONS.....	X
RESIDENT FORTRAN RUN-TIME PACKAGE.....	X
SHARED LOAD MODULES.....	X
TIME-OF-DAY CLOCK.....	X
WATCHDOG TIMERS FOR I/O DEVICES.....	X
LOAD FROM RANDOM ACCESS DEVICES.....	X
SYSTEM PROTECTED FROM UNDEBUGGED TASKS.	X
TASKS PROTECTED FROM EACH OTHER.....	X
ON-LINE PROGRAM PREPARATION.....	X
ON-LINE FILE CREATION, DESTRUCTION, AND EXTENSIBILITY.....	X
MEMORY PARTITION SIZES CHANGED ON-LINE.	N/A
MINIMUM MEMORY REQUIRED (BYTES).....	96K

FEATURE	MAX IV
MAXIMUM MEMORY SUPPORTED (BYTES).....	4Ø96K
MEMORY MAPPING.....	X
EXTENDED MEMORY AS GLOBAL COMMON.....	X
MULTIPLE BLOCKS OF GENERAL REGISTERS...	X
DIRECT MEMORY PROCESSORS REQUIRED.....	X
TTY OR CRT REQUIRED FOR OPERATOR.....	X
MULTIPLY/DIVIDE REQUIRED.....	X
DISC REQUIRED.....	X
MEMORY MANAGEMENT REQUIRED.....	X

LEGEND: o = no
X = yes
N/A = not applicable

APPENDIX B
STANDARD DEVICE AND CONTROLLER NAMES

This appendix describes the device names, disc controller names, and partition names that are considered standard for the MAX IV Operating System.

DEVICE TYPE	LOGICAL DEVICE NAME	STANDARD RECORD SIZE IN BYTES	CONTROLLER HANDLER NAMES
ASR-Teletypes (with integral paper-tape reader or special high-speed reader (CHALCO)	TYn (for keyboard/printer or CRT terminals)	74	TTx
	*PRn (for paper-tape reader -- high or low speed)	Ø(unlimited)	TTx
	**TPn (for ASR punch)	Ø(unlimited)	TTx
Paper-Tape Punch	PPn (for high-speed punch)	Ø(unlimited)	PPx
Card Reader	CRn (for card reader)	8Ø	CRx
Card Reader	CRn (for DMP card reader)	8Ø	DRx
Card Punch	CPn (for card punch)	8Ø	CPx
Line Printer	LPn (for DMP line printer)	134	DPx
Line Printer	LPn	134	LPx
Line Printer	LPn (centronics type matrix printer)	134	LCx
X-Y Plotters	XYn (for incremental plotters)	Ø(unlimited)	XYx
Raster Plotter/ Printers	VPn (for Versatec printer/ plotter)	Ø(plotter unlimited)/ 132(printer)	VPx
Null Device	NO (the "NO" device)	Ø(unlimited)	none
Magnetic Tape Devices	MTn	Ø(unlimited)	MTx
Communication Devices	Refer to the MAX IV SYSGEN, System Guide Manual listed in the Preface.		
CPU-CPU Link	CLx (for 482Ø non-MAXNET CPU Link)	Ø(unlimited)	CLx
Standard Link (4828 Controller)	OLx (Output Channel) and ILx (Input Channel) (for 4828 non-MAXNET Link only)	Ø(unlimited)	OLx ILx

Process I/O Devices	IDn (IOIS Digital I/O)	Ø(unlimited)	IDx
	IOn (IOIS Analog Output)	Ø(unlimited)	IOx
	IOn (IOIS Analog Output-Remote)	Ø(unlimited)	ROx
	MDn (MODAC Digital I/O)	Ø(unlimited)	MDx
	MOn (MODAC Analog Output)	Ø(unlimited)	MOx
	MOn (MODAC Analog Output-Remote)	Ø(unlimited)	ROx
	MIn (MODAC Analog Input)	Ø(unlimited)	MIx
	MIn (MODAC Analog Input-Remote)	Ø(unlimited)	RIx
	WSn (Wide-Range Solid State Analog-Input)	Ø(unlimited)	WSx
	HLn (High-Level Analog Input)	Ø(unlimited)	HLx
	WRn (Wide-Range Relay Analog Input)	Ø(unlimited)	WRx
	WRn (Wide-Range Relay Analog Input-Remote)	Ø(unlimited)	RIx
	DIn (MODAC or IOIS Digital Input-Remote)	Ø(unlimited)	RIx
	DOn (MODAC or IOIS Digital Output-Remote)	Ø(unlimited)	ROx
Fixed Disc Partitions (Floppy, Cartridge, or Multi-platter)	Any 3-character CAN code name that is unique to all device names. See next page.	200/256/ or unlimited	
Real Disc Files	See MAX IV FILE MANAGER, File Management Manual listed in the Preface.	User specifiable	

* Must be specified as second device of TT controller if optioned.
** Must be specified as third device of TT controller if optioned.

DISC CONTROLLER NAMES

Each disc can be partitioned into many logical devices. For the names of these devices see next section. The standard disc controller names are:

NAME	CONTROLLER
FLx	Floppy Disc
FHx	Fixed-Head Disc
CDx	Cartridge Moving-Head (Diablo & Wang)
CXx	Cartridge Moving-Head Double-Density (Wang)
MPx	Multi-Platter Moving-Head (ISS & Ampex)
MXx	Multi-Platter Moving-Head Double-Density (ISS and Ampex)
LDx	3330 Type Large Capacity (4138)
BMx	Memory Plus 4190 (bulk core)
LXx	Large Capacity Moving-Head Disc - Extended

DISC PARTITION NAMES AND SIZES

GLOBAL PARTITIONS

The following partitions are used to store sequential object element libraries that are installed and maintained by the Library Update program. Batch tasks usually share a single copy of each.

ALS System images in relocatable format for Stand-Alone Linking Loader (SAL).

AL4 Common elements (MAX IV) for system generation.

ALB Sequential object library of subroutines.

The following partitions are used to store (in a directorized form) source elements (Macros, Source Programs, Commands) that are installed and maintained by the Source Editor program. Batch tasks usually share a single one of each.

AMC Macro prototypes for MAX IV system generation.

AJC Command prototypes for JOB procedures ($DO) and Macro prototypes for FORTRAN compilations.

The following partitions are used to store quick-load object load modules (Tasks and Overlays) in directoried form that are installed and maintained by the Task/Overlay Cataloger program. These partitions are global to all tasks.

QSM Operator directive overlays and other nonresident system modules.

QBM Batch-processing task overlays (processors) and Job Control.

QLM Foreground tasks and overlays.

QLT Temporary load modules.

QTM Temporary programs under development.

TPØ Transient Partition definitions.

The following partitions are for direct access FORTRAN data storage and are installed and maintained by the Direct Access Maintenance Processor (DAMP) program. One of each of the following two data storage areas for Global Data Sets is sufficient if the data storage application is global to several tasks:

ADA
ADB

The following partitions are used for MAXNET IV only.

DLD Special stand-alone loaders.

DST MAXNET IV Satellite system images.

AMX MAXNET IV System Elements Library.

The following partitions are dedicated to specific system tasks.

QRO Partition used by the Roller task, must have an unlimited record size (Ø).

SPØ Partition used by the output spooler.

The following partition is supplied for custom Job Control directives:

BJM Custom Job Control Directives

PARTITIONS UNIQUE TO EACH TASK

Each batch-processing task should have the following unique files. The first character of the name (B) is shown for a particular task B. This character must be unique for each batch-processing task (C, D, ...).

The following partition is for source programs in directoried form that are installed and maintained by the Source Editor program.

BSL Source Library

The following partitions are scratch files and are used for transient data storage only.

BSC Main Scratch File.

BSA Small Secondary Scratch File.

BSB Big Secondary Scratch File.

BS Scratch File that overlays BSC, BSA, BSB.

BSX Extra Scratch File or Work File.
BSW

BJW Job procedure work file.

INDIVIDUAL PARTITION DESCRIPTIONS

The remainder of this Appendix describes the most common disc partitions. When more than one name is shown for a partition, the partition may be duplicated as necessary. Partition names beginning with B, C, D, ... are usually assigned exclusively to batch processing tasks B, C, D, ... Disc partitions that may be shared by several batch-processing tasks usually are given names

beginning with A. In most cases, the last two characters of a three-character partition name are equivalent to the task's logical file that is normally assigned to the partition.

Partition names beginning with X are usually accessed only by real-time tasks and are defined in such a way that they can only be assigned to global files or so that they are exclusive to a particular task.

Partition names beginning with Q are capable of being written with multi-sector records, such as are required for cataloging quick-load modules. Such partitions are assigned to global files if they must be accessed by any task.

Partition names beginning with any other letter are usually assigned exclusively to a task with a name that begins with that letter (for example, "S" for spooler task). None of the above rules need be followed if you choose to use either your own naming conventions or higher naming conventions.

ALS (Library of Systems and Stand-Alone Programs) is a sequentially accessed partition normally assigned (by default) to the batch-processing file LS. The contents of the partitions are installed and maintained by the Library Update program. The partition usually contains the following relocatable object programs separated by file marks ($$):

FILE	PRODUCT NO.	CONTENTS
File 0	-	Empty -- or user-selected program
File 1	600000-108	Disc/Tape Copy program (Stand-Alone)
File 2\	-	/Other stand-alone programs and
File 3/		\operating systems
File 4	-	MAX IV System Image (Edited or SYSBLOCK)
.		
.		
.		
File n	-	Other stand-alone programs and operating systems

The last element on the partition should be followed by a double file mark to indicate the end of useful information. The programs can be loaded by the Stand-Alone Linking Loader (SAL), which is normally installed in fillable form on tracks 0 and 1 of the System Residence transport (pack 0). The ALS partition normally starts with Track 2 of the system pack, the default starting track used by SAL. Users with limited disc space may delete this partition or include only enough room to hold a single operating system. All batch-processing tasks usually share a single copy of this file in a write-protected area of the disc.

AL4 (Library for MAX IV System Generation) is a sequentially accessed partition normally assigned to batch-processing file L4. The contents of the partition are installed and maintained by the Library Update program and usually include the following object subroutine elements that are unique to

MAX IV system generation. These elements must be installed in a single file with no intermediate file marks:

FILE	PRODUCT #	CONTENTS
File Ø	-	User-coded system generation elements (optional)
	61Ø3Ø4-999	MAX IV Final System Elements Library

A double file mark at the end of file Ø must mark the end of useful information on the partition. The optional user-coded elements must be located before the standard elements if they exist. The file is normally used by the Link Editor or Stand-Alone Linking Loader during the linking phase of system generation.

Users with only a small fixed-head disc may not have room for this partition and should use magnetic tape, paper tape, or card files instead.

All batch-processing tasks usually share a single copy of this file -- in a write-protected area of disc.

ALB
BLB
CLB
DLB
. (Library for Batch Processors) is a sequentially accessed partition normally assigned to batch file LB. This file is also known as the Sequential Object Library or the FORTRAN Subroutine Library. The contents of the partition are installed and maintained by the Library Update program and usually include the following object subroutine elements that are required for FORTRAN compilation or general program preparation linking. These elements must be installed in a single file with no intermediate file marks:

FILE	PRODUCT NO.	CONTENTS
File Ø	61Ø43Ø-ØØØ	MAX IV Utility Library
	61Ø411-ØØØ	MAX IV Math Library
	61Ø421-ØØØ	MAX IV FORTRAN Runtime Library
	61Ø435-ØØØ	MAX IV Executive Functions and Process I/O Library
	61Ø437-ØØØ	MAX IV FORTRAN Interface to File Manager
	61Ø9ØØ-ØØØ	MAX IV FORTRAN Interface to Intertask Communications System
	61Ø55Ø-ØØ1	MAX IV Load on Call

A double file mark at the end of File Ø must mark the end of useful information on the partition. The optional user-coded subroutines may be located anywhere on the library but should be located before any external routines that they may call if efficient linking is desired. The file is normally used by the Link Editor or Link Loader programs whenever a main program (FORTRAN or ASSEMBLY language coded) is being linked to its externally called subroutines. All batch-processing tasks usually share a single copy of this file in a write-protected area of disc.

If FORTRAN compilations are not to be performed on the system, then this partition (and file) may often be deleted.

AMC
BMC
CMC
DMC
. (Macros for MAX IV System Generation and FORTRAN macros) is a USL partition normally assigned to batch files MC and FM. This file is also known as the MAX IV Macro file. The contents of the partition are installed and maintained by the Source Editor program and include the following ASSEMBLY-language source elements, in compressed source format, that are required for MAX IV system generation. These elements are installed on a directoried USL file. The macro prototypes used by the MODCOMP CLASSIC FORTRAN compiler are also stored on this partition.

FILE	PRODUCT NO.	CONTENTS
File 0		Directory
File 1-N	-	User-coded custom prototypes
	610304-001	MAX IV Assembly Macros
	610304-0X2	MAX IV System Structure Macros (Parts 1, 2, and 3)
	610304-003	MAX IV REX Equates
	610304-005	MAX IV File Manager Equates and Macros
	611201-100	CLASSIC FORTRAN IV Macros

The optional user-coded prototypes may be cataloged with the standard elements. The file is normally used by the MACRO ASSEMBLER program during the assembly phase of MAX IV system generation.

FORTRAN MACROS

The file is normally used by the MACRO ASSEMBLER program during the assembly phase of FORTRAN compilations. Optional user-coded prototypes for other higher-level languages may be cataloged on the file.

Users with only a small fixed-head disc may not have room for this partition and should use a single sequential magnetic tape, paper tape, or card files instead.

AJC
BJC
CJC
DJC
(Job Control Procedures) is a USL partition normally assigned to batch file JC. This partition is sometimes known as the Job Procedure Library $DO Library. The contents of the partition are installed and maintained by the Source Editor program. It usually includes a compressed source format of simple Job Control statements.

These elements may also be installed on a single file with no intermediate file marks, in which case the most frequently used procedures should be installed first.

A double file mark at the end of File 0 must mark the end of useful information on the partition, if using a sequential partition.

If the $DO feature of Job Control is not used, the partition and its files may be deleted from the system disc.

If the user is building a system, the file and partition may be deleted or used for other purposes (such as storing system generation source decks).

QSM (Directoried Disc Partition for System Modules) is a randomly accessed disc partition, with a nonresident load module directory, normally assigned to GLOBAL file SM. The contents of the partition are installed and maintained by the Task/Overlay Cataloger program and usually include overlay programs used by the Operator Communication task and nonresident REX services. The partition must have a geometry of 1 and an unlimited record size so that it is capable of containing quick-load data format.

The directory is always accessed starting at disc Sector 0 of the partition. All object programs cataloged on the file are usually limited to 512 words when loaded into memory and must be cataloged in the Privileged mode. OC task directives must be biased at absolute #FE00 for loading into MAP 0; nonresident REX services must be cataloged in a relocatable format for relocation in MAP 0.

Whenever a hybrid operator task receives an operator directive, it searches the directory of the SM file for a program that has the three-character name that matches the three characters of the directive keyword. If found, the task loads the program (into a 512-word buffer) and executes it. This overlay may in turn load other overlays (one 512-word overlay at a time) until the directive function is executed.

QBM (Directoried Disc Partition for Batch-Processing Tasks) is a randomly accessed disc partition with a nonresident load module directory, normally assigned to the global file BM. The contents of the partition are installed and maintained by the Task/Overlay Cataloger program and usually contain overlay programs used by one or more batch-processing tasks, including the Job Control overlay.

The directory is always accessed starting at Sector 0 of the partition. All object programs cataloged on the file can be loaded and executed under Job Control as long as they do not exceed the size of the virtual addressing space. Numerous programs can be cataloged in the file, including the following necessary elements:

JOB CONTROL
TASK/OVERLAY CATALOGER
LIBRARY UPDATE
SOURCE EDITOR
SOURCE UPDATE
LINK EDITOR

Also, at least one version of the MACRO ASSEMBLER is necessary. This partition is not required unless a batch-processing task has been specified in the system version. The partition must have a geometry of 1 and an unlimited record size so that it is capable of containing quick-load data records.

QLM (Directoried Disc Partition for Real-Time Load Modules) is a randomly accessed disc partition, with a resident or nonresident load module directory, normally assigned to the global file LM. The contents of the partition are installed and maintained by the Task/Overlay Cataloger program. A resident directory is recommended for this file because of the amount of activity. The resident directory may be any size. If it is smaller than the disc-resident directory of this file, then the programs that fall outside of the resident directory range appear as if cataloged on a file with only a nonresident directory.

This partition requires a geometry of 1 and must have an unlimited record size so that quick-load data records can be written on it. If nonresident real-time tasks or overlays are not executed under the system, the file and partition may be deleted.

TPØ
TP1 This partition contains the disc definitions for transports configured by transient partition definitions for Basic I/O system volumes. The source for this is cataloged on the BSL partition.

DLD This partition is used for MAXNET software systems only and contains the MAXNET III/IV Stand-Alone Loader (SAL) and the SAL PREP programs.

DST This partition is used for MAXNET software systems only and is to be used for the fully edited object modules of the MAXNET IV Satellite System produced from the Satellite System Generation statements.

AMX This partition contains the object modules of the MAXNET IV System Elements Library for MAXNET systems only. Routines from this library are to be added to the routines on partition "AL4" and built into the MAXNET system by the system generation process.

ADA
BDA
CDA
DDA
.
.
.
XDA
.
.
. (Disc Partition for Privileged Global Data Bases) is a randomly accessed disc partition, normally assigned to the GLOBAL file DA. The partition is not directly accessed by the user's program, but is addressed indirectly by the FORTRAN define-file/direct-access extensions and data sets cataloged in the global RAD file directory. The partition is made assignable only to global files so that it can utilize areas of the disc not covered by write-protection hardware and also be given software-controlled access restrictions (including task-dependent write protection and read protection). The partition space is divided into data in the RAD directory by the DAMP program. This program can also be used to maintain

and display the contents of the data sets defined on the partition.

The ideal partition geometry depends upon the most frequently used data sets defined within it and how these data sets are used by applications programs. If the majority of accesses are sector-sized records, a geometry of 8 is recommended.

This partition is optional in MAX IV. It is useful where many real-time tasks must write into the same data base. Batch-processing tasks and undebugged tasks may be selectively given read access to this partition but should have write access prohibited.

ADB
BDB
CDB
DBD
.
.
XDB

(Other Data Set Partitions) are randomly accessed disc partitions, normally assigned to GLOBAL files DB, DC, ... Dn. These partitions are added when dedicated, unprivileged, or various read/write restricted data bases are necessary. The space on these partitions is assigned to the global RAD directory or to any local RAD directories by the DAMP program. This program can also be used to maintain and display the contents of these data sets.

These partitions are also added to a system when specific tasks (for example, batch-processing task B) have their own dedicated and permanent data bases. For example, a partition may be built that is not referenced by the global RAD file directory but that contains its own directory and data sets. The specific task would access such a data base by opening a local file with the name RAD (by assigning it to the global file already assigned to the dedicated partition). Such a task could then maintain its own data base and directory. These separate partitions are also suitable when write access is to be given to unprivileged tasks. Again, such accesses can be allowed by the global RAD file or a local RAD file.

ASL
BSL
CSL
DSL
.
.
.

(Disc Partition for Directoried Source Library) is a randomly accessed disc partition normally assigned to batch file SL. If many users can share a single source file, only one may be necessary. If some are to be write-protected for dedicated users, it may be necessary to include several partitions. The contents of each partition, including a special source are installed and maintained by the Source Editor program. This data is stored in a compressed ASCII format.

A partition geometry of 8 is required for fast discs, although a geometry of 4 will suffice for a cartridge movable-head pack. These partitions are optional in MAX IV.

ASC
BSC
CSC
DSC
.
.
.

(Main Scratch Partition) is a general-purpose (usually sequential access) partition, normally assigned to the batch task file SC. The partition is always assigned to a write-permitted area of the disc and should be dedicated and unique for each batch task configured. This partition is available to any batch overlay program for any purpose. It is used specifically by the Assembler, Macro Assembler, Assembler Cross Reference Program, and Link Editor programs for writing

data during Pass 1 so that they can automatically proceed with Pass 2 without further access of the original data file. It is also suitable as a temporary data set file for FORTRAN programs that use define-file direct-access extensions. In such cases, the Direct Access Maintenance Processor (DAMP) program can set up a suitable directory in the partition before the program is loaded, or data sets already referencing the partition can be in the global RAD directory.

ASA
BSA
CSA
DSA
. (Small Secondary Scratch Partition) is a general-purpose (usually sequential-access) partition, normally assigned to the batch task file SCA. The partition is always assigned to a write-permitted area of the disc and should be dedicated for batch task usage only. This partition is available to any batch overlay program for any purpose and is not used by any particular standard program. Common uses include the binary output file of assemblers and the binary input file of the Link Editor and Link Loader programs.

ASB
BSB
CSB
DSB
. (Big Secondary Scratch Partition) is a general-purpose (usually sequential access) partition, normally assigned to the batch file SCA. The partition is usually at least twice the size of partition DSA so that it can be used for specific jobs needing a larger partition. The partition is always assigned to a write permitted area of the disc and should be dedicated for batch task usage only. This partition is available to any batch overlay program for any purpose and is not used by any particular standard program. Common uses include the binary output file of the Link Editor or source output file of the FORTRAN Compiler.

AS
BS
CS
DS (Overlapping Scratch Partition) is a general-purpose disc partition that overlaps the physical media space already assigned to the scratch partitions xSC, xSA, and xSB, so it can be used only when these subpartitions are not in use.

The partition can be used by any batch program that needs one large scratch area. A common use of the partition is the compressing of large load module files that require a scratch file at least as large as the original file to be compressed. It might also be used for a large data base file if the main scratch file (xSC) is not large enough.

ASW
BSW
CSW
DSW
. (Extra Scratch/Work File) is a general-purpose disc partition, normally assigned to the batch task file WRK. The partition is always assigned to a write-permitted area of the disc and should be dedicated for batch task usage only. This partition is usually required in a system only if the Source Editor program is used. If many applications require the very large scratch file (DS), then this file is suitable as a second scratch file.

AJW
BJW
CJW
CJW
DJW (Job Control Work File) is a sequentially accessed disc partition, normally assigned to the batch task file JW. This partition should only be used by the Job Control Overlay Program. Whenever Job Control receives a $DO command, it searches the JC file (partition xJC) for the specified

. procedure and then copy it in uncompressed source format to
. the JW file. It then temporarily assigns the CI (Control
. Input) file to JW. The size of the partition must be based on the largest $DO procedure to be expanded, (one sector of JW is used for each statement of the expanded procedure). Also consider the effects of nesting $DO's.

SPn (Partitions for Spooling of Imaginary Devices) are assigned exclusively to a spooling task such as "S". A spooling partition must be specified for each imaginary "printer" device defined in the Device table shown at the beginning of the Appendix.

APPENDIX C
STANDARD GLOBAL LOGICAL FILE NAMES

This appendix identifies the standard MAX IV global logical file names. The names shown in the following table are system files usable by any task as long as limitations have not been imposed such as task influence limit or exclusivity. (A detailed description of each file follows the chart.) Refer to Appendix D for files names used in batch processing.

GLOBAL FILE NAME	NAME AND PURPOSE	ASSIGNMENT
CO	Computer-to-Operator file for printing all messages that should be read by the "system" operator.	TYn (or any printing device)
DO	Diagnostic Output file for printing diagnostic dumps and error logs.	TYn, LPn (or any printing device)

The following global disc files are also assumed to exist in a system.

LM	Load Module file for real-time tasks and overlays cataloged in quick-load format.	(any disc partition)
BM	Batch Module file for batch-processing tasks and overlays cataloged in quick-load format.	(any disc partition)
SM	System Module file for nonresident system elements and operator directive overlays cataloged in quick-load format.	(any disc partition)
RO	Roll-out file used by the ROL task for checkpointing low priority tasks when their memory is needed by high priority tasks.	(any disc partition)
TM	Temporary Load Module file. Used for cataloging tasks and overlays in quick-load format during development.	(any disc partition)
TPD	Transient Disc Partition definitions	(any disc partition)

OPERATOR COMMUNICATION FILES (OC TASK FILES):

OC	Operator-to-Computer file for reading operator directives.	TYn (or any input device)
CO	Computer-to-Operator file for the OC task.	TYn (or any printing device

DETAILED DESCRIPTION OF FILES

OC The OC (Operator Communications Input) file must be included in all systems. This file is used by the Operator Communication task to accept all input directives for the system operator. Normal file assignments and default file assignments must be directed to a terminal input device such as:

- o console (TY)
- o remote teletypewriter
- o alphanumeric CRT (TYn).

The normal assignment of this file is returned to the default device assignment whenever the CONSOLE INTERRUPT switch is pressed on the computer's control panel. The READ and WRITE access can optionally be restricted to the Operator Communication task.

CO The CO (Communications Output) file must be included both globally and for the OC task in all systems. This file is used by the Operator Communication task to type all device or task error messages. The Global CO file is used to type all information reports to the operator. The Global CO file is also the file used by the REX MESSAGE service (REX, #1Ø) to output messages to the operator from each task level. Therefore, the WRITE access should not be restricted. The file must have both its normal and default assignments directed to a printing device, usually a terminal such as:

- o console (TY)
- o remote teletypewriter
- o CRT (TYn).

When the CONSOLE INTERRUPT switch is pressed on the control panel the default assignment replaces the normal assignment in the OC's CO file.

SM The SM (System Load Module) file is required in all systems. This file should have its normal and default assignments should be assigned to a dedicated disc partition. The file contains nonresident executive service modules and OC directives.

LM The LM (Load Module) file is used to load the most important nonresident real-time tasks and/or overlays. The LM file is selected by default at runtime if no file is specified when the user activates, establishes, or connects tasks, or when program overlays are loaded at real-time task levels.

If a resident directory is specified for this file, the file maintains the most important programs in the resident directory (ensuring faster access), and the remainder of the programs are maintained in the nonresident directory. Other files can be used for real-time tasks. Such files can have any practical number of tasks and overlays cataloged within them. The file must always be assigned to a disc partition.

BM The BM (Batch-Processing Load Module) file is a load module file with a resident or nonresident directory. BM is the default file used to load all batch overlay programs when no file is specified. It is also used as an overflow real-time task file should LM be full. The BM file should have both its normal and default assignments directed to a disc partition (usually BM). The BM file can actually be made a local batch task file instead of a global file. In this case, it could not serve as an overflow file for cataloging additional real-time tasks.

TM The TM (Temporary Load Modules) file is a general purpose load module file with a nonresident directory. Use the TM file when large numbers of tasks are undergoing program development requiring frequent replacement. This file is usually assigned to a WRITE permitted area of disc so that new tasks and overlays can be freely cataloged without the need to unlock critical WRITE protected files. No READ/WRITE access restrictions are required unless software controlled WRITE protection is desired. (In this case, the file usually has the disc partition QTM assigned to it.) Like BM, this file can be local to a batch-processing task instead of being made global.

TPD Transient Disc Partition Definitions are independently assembled in the SYSGEN process. The structures generated by the definitions are designed to be actively linked into the system at start-up. The structures are Logical Device Tables (LDT) and exist as non-standard binary structures on the disc file TPD.

DO The DO (Diagnostic Output) file is used by the optional DUMP service. If the DUMP service is specified, the user should include this file and assign it to a high speed printing device.

RAD The RAD (Random Access Directory) file is optional. Add only when using the direct-access/define file extensions of the FORTRAN Runtime Package. This extension permits disc data-set areas to be subdivided into many logical subpartitions. If a global RAD file is included, then these data sets can be made into a global data base that is referenced by many tasks.

A RAD file can also be included as a local TASK file in which case all data-sets defined in its directory are local to that task. READ and WRITE access to the global RAD file can be restricted but this is unnecessary if the partition is assigned to a WRITE protected area of disc.

DA
DB
. DA and DB (Data Set Files) are optional. When the RAD file is assigned to a partition in the WRITE protected area of disc and contains only a directory and READ only data sets, then other partitions must be included (they are referenced by the RAD directory) to support WRITE permitted data sets.

If several restriction levels exist, or if dedicated data bases exist, several data-set files can be required. These files are normally assigned to disc partitions ADA, ADB, ... These files need not be made global unless access is to be restricted. If they are to be used exclusively by one task, they can also be local files.

RO The RO (Roller) file is required in any MAX IV system that has the ROL task installed. The file must be assigned to a disc file capable of being written to in a quick-load format. The disc space for the file must be suitably sized to contain the bodies of all tasks that might be capable of being rolled-out to accommodate higher priority tasks that need additional memory resources.

APPENDIX D
MAX IV STANDARD BATCH-PROCESSING LOGICAL FILES

This appendix lists the standard logical files for batch-processing operations under the MAX IV Operating System. The appendix begins with a chart that provides an overview. Following the chart is a detailed description of each file. (Refer to Appendix C for explanations of global logical files.)

FILE	NAME	TYPICAL DEVICE ASSIGNMENTS	SOME USES	WRITE PROTECTED
Command Input	CI	CRn or TYn	Read JOB command directives	No
Alternate Command Input	AI	TYn or CRn	Read JOB command directives	No
Listing Output	LO	LPn or TYn (printer)	Print reports and status	No
Source Input	SI	CRn or TYn	Read language processor input statements	No
Source Output	SO	BSB (SCB)	Compiler output	No
Binary Input	BI	BSA (SCA)	Loader and editor inputs	No
Binary Output	BO	BSA (SCA)	Assembler and editor outputs	No
Library (Subroutines)	LB	ALB	FORTRAN subroutines and routine elements	Yes
FORTRAN Macros	FM	AMC	Compilations	Yes
Job Control Procedures	JC	AJC	Command procedure prototypes ($DO)	Yes
Job Control Custom Directives	JM	BJM	Custom Directives for Job Control	No
Library of Systems and Stand-Alone Programs	LS	ALS or hard-copy device	Core image of operating system	Yes
Library of MAX IV System Elements	L4	AL4 or MTn	System generation linking phase	Yes *
MAX IV System Generation Macros	MC	AMC or MTn	System generation assembly phase	Yes *

File Manager Macros and Equates	FMM	AMC	System generation assembly phase or assemblies	Yes *

* For a moving-head cartridge disc (DIABLO) and a fixed-head nonremovable disc, these files are not protected.

The following files must be assigned to unique devices for each batch-processing task. The device assignments are shown only for the first batch-processing task installed in a system. This task (B) has the character "B" prefixed to all disc partitions used exclusively by this task.

FILE	NAME	ASSIGN-MENTS	SOME USES	PROTECTED
Main Scratch	SC	BSC	Automatic second pass	No
Small Secondary Scratch	SCA	BSA	For two pass programs utility	No
Big Secondary Scratch	SCB	BSB	Utility	No
Job Work Scratch	JW	BJW	Expanded JOB procedures	No
User's Source Library	USL	BSL	For Source Editor	SOFTWARE
Work Scratch or Extra Scratch	{WRK} {SCX} {SCC}	BSW or BSX	Utility or for Source Editor random mode updating	No
FORTRAN Operator Messages	1	((CO))	FORTRAN	No
FORTRAN Symbolic Input	2	(SI)	FORTRAN	No
FORTRAN Listing Output	3	(LO)	FORTRAN	No

DETAILED FILE DESCRIPTIONS

The following files are required by most batch-processing tasks. If more than one such task exists, the names can be duplicated in each such task's File Assignment Table (FAT). Such duplicate files can be assigned to common or unique devices, depending upon whether sharing is required or unique files are required. Scratch files (that can be written to) are generally assigned to unique devices. Library files (containing macros, subroutines, whole programs, and tasks) can be assigned to shared partitions if they are accessed in a READ only mode.

CI The CI (Control Input) file must be included in each batch task's FAT. The CI is used by Job Control and by many batch processors to read control directives. Both normal and default assignments can be made to a teletypewriter.

LO The LO (Listing Output) file must be included in each batch task's file list. This file is used by Job Control and other batch processors as a listing (printing) file. All Job Control statements received by file CI are listed on LO. Both normal and default assignments are usually made to a high speed printer or teletypewriter.

SI The SI (Source Input) file can be included in any batch task's FAT if a standard source input device can be chosen. Otherwise it can be opened from a vacant FAT entry by an $ASSIGN statement or as part of a $DO procedure prototype. This file is used by all language processors (Assemblers, Compilers) and general source maintenance programs to read source (symbolic) statements and comments. Both normal and default assignments can be made to any device from which source most often originates.

SO The SO (Source Output) file can be included in any batch task's FAT if a standard source output device can be chosen. It can be opened from a vacant FAT entry by an $ASSIGN statement or as part of a $DO procedure prototype. This file is used by the FORTRAN Compiler and general source maintenance programs to write source. It is also used as an input file for Macro prototype insertion during system generation. Both the normal and default assignments can be made to any output device (usually a disc scratch file) or to any input device if the FORTRAN compiler is used less frequently than MACRO insertion.

BI The BI (Binary Input) file can be included in any batch task's FAT if a standard binary (object) input device can be identified. It can be opened from a vacant FAT entry by an $ASSIGN statement or as part of a $DO prototype. This file is used by the Editors, Loaders, and general object maintenance programs to read object data. Both normal and default assignments are usually made to a disc partition or any other input device normally used to process object programs.

BO The BO (Binary Output) file can be included in any batch task's FAT if a standard binary (object) output device can be identified. It can be opened from a vacant FAT entry by an $ASSIGN statement or as part of a $DO procedure prototype. The BO file is used by the Assemblers, Editors, and general object maintenance programs to write object data. Both normal and default assignments are usually made to a disc partition, paper tape punch, or any other output device normally used to prepare object programs.

LB The LB (Library) file can be included in any batch task's FAT if a standard object library device can be identified. If FORTRAN programs are prepared under your system, this file must be included. This file is used by the Link Editor or Link Loader to locate subroutines that have been called by an unedited main program, particularly a FORTRAN program. Both normal and default assignments are usually made to a disc partition, paper tape, or any other input device where the library object routines are located. A WRITE protected area of disc is used.

SC The SC (Scratch) file can be included in any batch task's FAT if a main scratch device can be identified. The SC file is used to write information during the first pass of Assemblers or the Link Editor, so that the information can be automatically read during the second pass of these programs. Such a device must be bi-directional (READ and WRITE capability) and rewindable without operator intervention. Both normal and default assignments are usually made to a disc partition or magnetic tape transport. If the task's system option SC is specified, this file should be permanently assigned in the batch task's FAT. Each batch task must have a unique device assignment for SC.

SCA
SCB These secondary scratch files hold intermediate results for any series of batch task operations. For example, the files BO, BI, SO, and SI are often assigned to these files in Job Control procedures so that such procedures can be made independent of device (partition) names. Each batch task must have a unique device assignment for these files.

WRK The WRK file should be included as an extra scratch file for SCX (the Source Editor). Each batch task must have a unique device assigned to this file. The file is used to expand compressed source.

FM The FM (FORTRAN Macro) file can be included in the batch task's FAT if a standard device suitable for storing Macro prototypes can be identified. FORTRAN compilations require the presence of this file. Both normal and default assignments are usually made to a disc partition in a WRITE protected area of disc or other rewindable device.

JC The JC (Job Control) file is optional but must be added to each batch task FAT if the Job Control procedures ($DO). Job Control uses this file to store symbolic procedures. Both normal and default assignments are usually made to a disc partition in a WRITE protected area of disc.

JM The JM (Job Control Custom Directives) file is optional. It is provided for custom written Job Control Directives. Assignment can be to one disc partition for all users or individual assignments depending on the applications. This file is searched in addition to JC to invoke Job Control procedures.

JW The JW (Job Work) file is optional but must be included in each batch task's FAT if the JC and JM files are specified. This file is used by Job Control as a scratch file when it is processing and expanding procedures ($DO). Both normal and default assignments are usually made to a disc partition that is in a WRITE permitted area of the disc. Each batch task's JW file must have a unique device assignment.

LS The LS (Library of Systems and Stand-Alone Programs) file is optional. It should be assigned to the device (usually disc partition) to which the current MAX IV Operating System image(s) and other stand-alone programs are to be stored.

L4 The L4 (Library of MAX IV System Generation Elements) file is optional and normally assigned to the device where all unique MAX IV object elements are located. This is a dedicated disc partition if disc pack space is sufficient to justify these elements using permanent disc spaces.

USL The USL (User Source Library) file is optional. It should be assigned to the disc device containing the source library.

MC The MC (System Generation Macros for MAX IV) file is optional and usually assigned to the device where all MAX IV system-generation Macro prototypes are located, normally a dedicated disc partition.

FMM The FMM (File Manager Macro) file is optional. It is normally assigned to the MC file and is used by various File Manager subroutines for inserting macros, equates, and so on.

AI . . Other batch files might be required for particular programs but can be opened at run-time from VACANT files. At least five vacant files should be available normally in the batch-processing FAT for this purpose.

If using FORTRAN programs primarily, standard sequential FORTRAN files (1, 2, 3, ..., 999) can be established permanently. Otherwise, they can always be opened from the vacant FAT for each job. Compare the advantages of having permanent batch processing files (using 6-8 words each), with the disadvantage of having to assign them for each job that uses them. The following files are normally used by FORTRAN programs:

File	Use
1	Operator message file
2	Source input file
3	Listing output file

APPENDIX E
ERROR MESSAGES AND RECOVERY PROCEDURES

Appendix E explains the error messages and the recovery procedures, where appropriate, that are provided by the MAX IV Operating System. The appendix is organized into the following sections:

1. System Initialization
2. I/O Device Offline Messages
3. No Volume Mounted Message
4. Task Abort Messages
5. Loader Error Messages
6. Parity Error Messages

SYSTEM INITIALIZATION

During system initialization all memory in the system is checked. If any location fails, that page is marked as not available to the system. A message indicating the number of unavailable pages, if any, is output to the operator during startup. The message format is:

```
BAD PAGE COUNT = xxx
```

If no errors are found, no message is output.

I/O DEVICE OFFLINE MESSAGES

These messages are printed on the system operator's terminal whenever the Basic I/O System has attempted system recovery for an I/O device operation that found errors, and all attempts of automatic recovery by the handler have failed. 'xxx' is replaced by the name of the device.

MESSAGE	CODE	REASON
!xxx-OFFLINE	OVF(status)	Data sync error; overflow or underflow
!xxx-OFFLINE	PAR "	Device data check error or parity error
!xxx-OFFLINE	INO "	Device inoperable or Watch-dog timer expiration
!xxx-OFFLINE	MPE "	Memory parity during DMP I/O operation or programmer violated virtual addressing space rules
!xxx-OFFLINE	LOK "	Attempted WRITE in protected area of medium
!xxx-OFFLINE	CIG "	Character in gap -- unrecoverable media failure (for example, loss of disc sector formatting (prep))
!xxx-OFFLINE	OTH "	Device-dependent error
!xxx-OFFLINE	BLK "	DMP (Direct Memory Processor) failure to transmit complete block or other device-dependent reason
!xxx-OFFLINE	OFF "	Operation that caused failure has been removed from queue by abort service

NOTE: Special attention is called to the MPE error. This type of error can either mean that hardware actual memory failures have occurred -- or that the program that scheduled the I/O operation has specified a buffer or data chain element that falls out of the valid virtual addressing range of the program. For example, trying to read data from an I/O device into a memory area that has been given READ only access rights causes MPE errors. Attempting a READ or a WRITE to/from a page to which no actual memory is allocated also causes MPE errors.

Recovery procedure from many of the above errors is device dependent. In this case, the physical device might have error indicators lit that help the operator to uniquely define the problem. Specific device recovery procedures, are explained in the MAX IV OPERATOR COMMUNICATIONS, System Guide Manual listed in the Preface.

When the device has been made ready, the device can be returned to operating state by one of the following operator communications responses:

```
/ON xxx
[/task]/ON lfn
```

The above entry places device xxx online and causes the previous operation(s) to be repeated. If the device continues to produce errors, the system operator can abort the task or force continuation as if the operation had not been in error.

The following directive causes device 'xxx' to be put online and the I/O system continues to operate as if the operation was completed successfully without error. When using the following, the program may use inaccurate data.

```
/ON xxx NO
[/task]/ON lfn [NO]
```

The parameter NO indicates that NO RETRY will be attempted.

NO VOLUME MOUNTED MESSAGE

This message is printed on the system console whenever an I/O operation is queued to a transport that is logically not ready (either no volume is mounted, the transport is not up to speed, or there is a mis-match between the responsibility of the transport (File Manager or BIOS) and the type of volume mounted (File Manager or BIOS)).

```
!xxx NVM?/task/overlay ('xxx' is the logical device name)
```

TASK ABORT MESSAGES

Some errors are so serious (or the programmer has chosen not to deal with them) that the task program that caused the error is abnormally terminated. When this occurs, a message in the following format is logged on the system operator's terminal:

```
!ABORT( why what where )/taskname/overlayname
```

The "why" code is a three-character word that describes the reason for the abort. These codes are listed below in alphabetical order. For some abort messages a "what" code is also printed. This code gives more information about the conditions of the abort. A hexadecimal "where" number is always printed, and usually identifies the virtual address within the aborted task program where the Executive service that resulted in an abort condition was called by the program. If an asterisk (*) follows this number, then the address refers to a location mapped in the operating system nucleus (MAP Ø) instead of in the addressing space of the aborted task.

ABORT "WHY" CODES IN ALPHABETICAL ORDER

WHY	WHAT	REASON AND RECOVERY
AFD	CHN	A privileged task/overlay has attempted a data chained I/O operation to an Asynchronous Floppy Disc device. Data chaining is not supported on AFD devices.
AFD	DMI	A task/overlay has attempted to transfer more than 4Ø96 bytes of data to/from an Asynchronous Floppy Disc device using the Direct Memory Interface (CP). A maximum of 4Ø96 bytes can be transferred to/from an AFD device when using the DMI.
ASC	fna	The ASSIGN service was called to close the logical file "fna" (by specifying a device name of zero) but "fna" was an unknown logical file.
ASG	xxx	The ASSIGN service was called to associate a logical file with some unknown device (xxx) or another unknown logical file (xxx). The name of a valid device or logical file must be supplied to the service.
BFR		SDLC/HDLC buffer size(s) are incompatible with device param eter.
CHN		Illegal chain list location specification while data chaining.
CM		The Link Loader program has detected an error in the object program it is loading. The common blocks in this program do not have matching

sizes, or the Assembler-produced illegal object code. You must re-assemble the program and/or catalog it.

CON		The CONNECT service has been called with an illegal argument. The "where" address points to the location of the call in the task addressing space.
CK		The Link Loader or Sequential Loader has found a checksum error, a directory error, or some other format error in the object program being loaded. This program must be re-installed on the specified file.
COP	OC	Computer operator abort.
CPF	xxx	The ASSIGN service was called to close-out a permanent logical file entry (xxx). Only non-permanent logical files can be returned to the vacant state.
DCI	OC	The operator has used the DCI OC Directive to change the assignment of the attentive task's CI file and to abort that task.
EF		The Link Loader or Sequential Loader has found a file mark on the specified sequential file instead of the desired program. The program must be installed before the first file mark.
ERC	xxx	Illegal error recovery code specified in UFT.
FAL	fna	The ASSIGN service was called to associate a logical file with another logical file (fna) that would have created a file-to-file loop. Loops are either immediate or anticipated. An immediate loop is checked for by following the current assignments in the file-to-file chain. An anticipated loop is checked for by assuming that some future ASSIGN of the same logical file could move the default assignment to the current assignment and cause an immediate loop.
FC		The Link Loader has found a function code error in the object program being loaded. The program must be re-installed on the specified file, or the correct file must be specified.
FCN		The Sequential Loader has found an illegal function code or illegal program name in the specified object program. The program must be re-installed or the correct file specified.
FER		FORTRAN error (xxx), the reason codes may be found in the MAX IV FORTRAN IV Language Reference Manual listed in the Preface.

FIL		The JOB CONTROL program has found an error in a $MOVE or $DO statement. Check statement deck or procedure carefully for errors.
hh	FM	The File Manager has aborted the task due to REX error hexadecimal "hh".
FME		File Manager Job Control has aborted the task due to an error return from a File Manager REX call.
FMF	fna	The ASSIGN service made an illegal attempt to assign a file (fna) to a File Manager File Assign Table.
FMG	nn	A File Manager REX service has completed and has returned the error code nn. The Abort-on-Error option was specified when the service was called.
FNO	xxx	File Manager file not open.
FPI	xxx	The disc device assigned to the specified logical file (xxx) is being addressed with an illegal File Position Index. A physical record within the valid range of the device must be used.
FPZ	fna	The ASSIGN service was called to clear either the current or default assignment for the logical file "fna" (specified in the calling UFT) but there exists some other logical file(s) in the user's File Assign Table list whose current or default assignment points to "fna". The current or default assignment of the other logical file(s) must be cleared or assigned elsewhere in order to clear the current or default assignment of "fna".
FSE	xxx	The device assigned to the specified logical file (xxx) has been accessed more than 65536 times on a single file search operation.
GEO	xxx	The disc device assigned to the specified logical file (xxx) is being addressed in FPI mode for a multi-sector transfer and the geometry of the device is not 1.
GO		The Job Control program has been requested to load a new system processor and a job error was signaled during the execution or the previous system processor, and the GO option has not been specified. Correct the cause of error and re-run the job, or specify the GO option to permit processing with errors and then re-run the job.

HI		The Sequential Loader or Link Loader is attempting to load the requested program over the loader itself or the symbol table. There is no recovery unless the program is made smaller or loading starts at a lower address.
IFN	xxx	An I/O service call was made that referenced a logical file (xxx) that does not exist. This logical file must be defined for the calling task or must be added to the global logical file assignment list of the X task.
III	CHN	MAX III type data chaining not allowed.
ILD		The Task/Overlay Cataloger has found an illegal directive while interpreting commands from other than an interactive terminal.
ILV		The requested interrupt level to be connected is not legal.
IOB		An I/O WRITE call was made that is using the Standard Binary mode, and the data buffer does not contain the required standard header in the first two (2) words of the buffer. The program must be modified to provide this header.
ISR	ARV	An unprivileged caller tried to insert a privileged shared region or a caller caused an access rights violation when trying to insert a shared region.
JCE		The Job Control program has found an illegal statement. The statement deck or procedure must be corrected.
KIL	INT	The specified task has been aborted by a task-scheduling interrupt.
KIL	TIM	The specified task has been aborted by a task-scheduling timer.
LOC		The Job Control program has found an error while processing a $LOCATE statement.
LOK		A disc partition or tape transport has a hardware lock condition in effect (write ring removed) and a WRITE was attempted by the calling task.
MAP	LEN	The I/O system was called with a buffer address or size that was not fully within the allocated memory space of the calling task.
MEM	PAR	An unrecoverable parity error occurred while executing this task.

MSO		The task's main push stack overflowed.
NVE	xxx	The ASSIGN service was given a logical file name that could not be found in the file assign table, and there were no vacant entries in this table. Re-install the task with more vacant files or add to global list.
PGM		The Job Control program cannot find the program requested by the latest $EXECUTE statement. Specify correct program name or correct file name.
PIO	MOD	Illegal Process I/O mode of operation requested.
PIO	GU	Illegal Process I/O group/unit number.
PRO		The calling task has executed a privileged instruction or has violated the access-rights of its assigned addressing space. The "where" address points to the region of failure in the aborted task - unless it is followed by an asterisk (*) in which case this address is in Map Ø. The program must be corrected so that it will not make the violation.
PRV	CHN	An unprivileged task has called an I/O system service requesting data chaining.
PRV	DMR	An unprivileged task issued a Diagnostic mode request.
PRV	DPI	An unprivileged task has attempted to use an I/O system service that is reserved for privileged task use. Using absolute disc address coordinates is not permitted by unprivileged tasks.
PRV	IOW	An unprivileged task called the I/O WAIT service specifying a UFT list in Map Ø.
PRV	MAP	Illegal use of maps.
PRV	OTF	Unprivileged task tried to use another task's File Assign Table.
PRV	PAK	An unpriviledged task has attempted to take exclusive use of an entire pack.
PRV	TER	Illegal use of TERMINATE service.
PSO		A task's PS/PR push stack overflowed.
PXU	xxx	Requested logical device has been taken for permanent exclusive use.

REX		The calling task has executed an unimplemented instruction or REX service for which no processing routine or simulation was supplied at system generation time - and local return from such errors has not been specified by the task.
SEC	xxx	An I/O request was made to an opened File Manager file and the user's permissions do not allow the requested operation, for example, a write to a read-only file. (The "xxx" represents the associated logical file name.)
SOF		The calling task has an underflow/overflow of the stack reserved for it in its TCB. The task must be corrected or re-installed with a larger stack.
SQ		The Sequential Loader or Link Loader has found an object record sequence error while loading the requested program. The program must be re-installed or the correct file must be specified.
STC	xxx	An I/O request was made to a random-access device and the user's positioning request has been resolved to an absolute device positioning in which either the sector, track, or cylinder is outside the physical limits of the transport. (The "xxx" is the associated logical file name.)
TCB	TOC	EXT page room abort. 8 pages filled.
TIM		Task's CPU utilization time limit exceeded.
TOC		The Task/Overlay Cataloger has found an illegal TCB size. The task must be recataloged to get rid of the error.
TNQ		A privileged task has called the I/O system and requested the use of another task's files and the other task is not currently active. The task must be established or resident or active at the time of the call.
TYP	xxx	Illegal File Manager file type.
UAS	xxx	An I/O system service was called that referenced a logical file (xxx) that did not have a device assigned to it. The file must be assigned to a device or other logical file that is ultimately assigned to a device.
UMF	LOD	See next section of this appendix titled "LOADER ERROR MESSAGES"
UNC		A request was made to thaw an unconnected interrupt level.

UST	NEG	User-specified timer was negative.
XT		The Link Loader has found an illegal external reference. The program being loaded must be re-installed or the specified file must be correct.
ZAP		A MAX III compatible REX service has been called that provided for the address of an error processing routine and none was supplied by the programmer and an error has been detected by the service.
ZAP	CAD	Allocate failed (see above).
ZAP	LOD	Loader error (see above).

NOTE: If the ABORT code is not found in this appendix, then refer to the documentation of the program being executed when the abort occurred as specified by "overlayname" on the ABORT message. If this program is a system batch processor, refer to the appropriate manual. For File Manager messages, refer to the File Manager File Management Manual listed in the Preface.

LOADER ERROR MESSAGES

Regardless of whether or not the individual error messages of the Module Loader are suppressed or permitted, an erroneous load operation will end with an ABORT message with the reason code UMF. If individual error messages are requested when the loading operation is initiated, then one or more such messages, in the following form, are printed on the system operator's terminal prior to the abort message, as an aid in determining the specific cause of the error.

LOAD ERROR AT #nnnn - lec /taskname/overlayname

The "#nnnn" is the instruction number of the LPR instruction as indicated in the Map produced by TOC and "lec" is a loader error code. These error codes are listed in alphabetic order below. Additional information can be obtained by reading the Quick Load Module chapter of this manual. Recovery, in almost all cases, requires recataloging the module or specifying the correct load module file name.

CODE	MEANING
ALF	Allocate failure. The load program requests memory allocation in some region that does not lie wholly within the virtual addressing space defined by the referenced map.
ANO	Attempted nucleus overlay. The load program requests loading in Map Ø and into a region allocated to the MAX IV nucleus.

ARV	Access right modification failure. The load program is attempting to change access rights in a region in which no such change is permitted.
CPF	Create private shared region failure. The load program is attempting to create a shared region in a area where it is not permitted.
DEF	Deallocation failure. The load program is attempting to deallocate memory in a region where it is not permitted.
DVF	Delete named variable failure.
DVI	Delete named variable incompatibility.
ERF	Erase shared region failure.
FAD	FAT entry definition failure.
FAE	File assignment error. If a file assignment error occurs, the following message describes the assignment error. LFN=xxx NOW=yyy DEF=zzz "xxx" specifies the logical file to be assigned, "yyy" is the new assignment being made and "zzz" is the default assignment being made. "yyy" or "zzz" is in error.
FAR	FAT entry replacement failure.
ICF	Insert copiable module failure.
IGF	Insert global shared region failure.
IMF	Insert shared load module failure.
IPF	Insert private shared region failure.
MAF	Memory allocation failed.
MRF	Module relocation failure.
NME	No module entry point.
NSF	No symbiont PDT found.
NTM	Not a two map task.
POI	Set program option incompatibility.
PRE	TOCLOCK prevent. Attempt to load a shared load module or segmented program while a squeeze of the load module file was in progress.

PSC	Privileged state conflict.
RNL	Region not loadable.
SCF	Set area constant failure.
SOI	Set system option incompatibility.
SNV	Set named variable error.
SVS	Set variable space.
UOC	Undefined loader operation code.

PARITY ERROR MESSAGES

These messages are printed on the system operator's terminal after a parity error has occurred. Three different messages are produced, all with the common header:

```
MEMORY PARITY ERROR OCCURRED
```

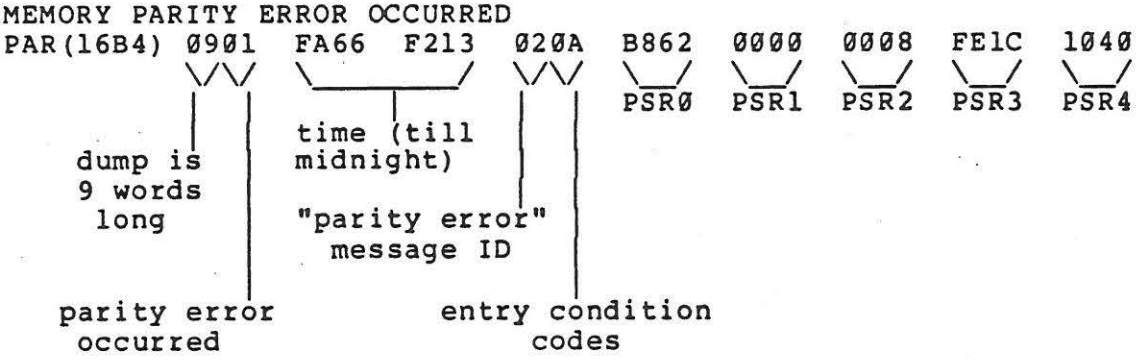
The second line of each message defines the message type and contents. It is a hexadecimal dump of memory containing the message contents. There are three messages that are dumped:

(1) Rate error message (SID=00): this message is dumped when the parity error rate has exceeded (normally) 10 errors per second.

(2) Correction on message (SID=01): this message is dumped when the parity error rate exceeds (normally) 5 errors per second on a single memory plane.

(3) Parity error message (SID=02): this message is dumped each time a parity error occurs (unless the error rate exceeds 10 per second).

Example:


MEMORY PARITY ERROR OCCURRED
PAR(16B4) 0901 FA66 F213 020A B862 0000 0008 FE1C 1040
PSR0 PSR1 PSR2 PSR3 PSR4
dump is 9 words long
time (till midnight)
"parity error" message ID
parity error occurred
entry condition codes

APPENDIX F
FAULT ISOLATION PROCEDURE

This appendix explains the MAX IV fault isolation procedure, which is used when MAX IV fails and falls into a non-recoverable situation. The procedure determines where the problem exists.

CLASSIC:

1. Press the HALT switch.
2. Turn the panel key to ENABLE.
3. Reset all switches. ADDRESS display holds PR.
4. Set CPU and PSW. DATA display holds PSW.
5. Set CPU and interrupts. DATA display holds ACTIVE INTERRUPTS.

6a. On 7860 and 7870 set CPU, interrupts and EMCR. DATA display holds REQUESTING INTERRUPTS.

6b. On 7830 set CPU, interrupts, and ISTK. DATA display holds REQUESTING INTERRUPTS.

7. Dump the system.

To DUMP a MAX IV SYSTEM that has the SYSBUG DUMP option configured:

1. Turn on printing device.
2. HALT CPU.
3. Master Clear CPU.
4. Set data switches to #0400.
5. On a CLASSIC press P/MA switch.
6. Press the RUN switch.
7. Wait for completion of printing.

DATA ANALYSIS: If Bits 0-2 of the PSW are all zero, find the PR address at which you halted in your LINK EDIT MAP. Record the name and starting address of the module in which you halted and the address at which you halted.

APPENDIX G
OVERLAY AND BUFFER CONTROL BLOCKS

This appendix shows the structure of the Overlay Control Block and the Buffer Control Block.

OVERLAY CONTROL BLOCK (OCB)

```
 1  * OCB EQUATES
 2  OCBSTA EQU 0          WORD 0       STATUS
 3  OCBADR EQU 1          WORD 1       VIRTUAL ADDRESS OF OVERLAY
 4  OCBRET EQU 2          WORD 2       ENTRY ADDRESS INTO OPA
 5  OCBBCB EQU 3          WORD 3       POINTER TO BCB
 6  OCBSPT EQU 4          WORDS 4-8    STACK POINTER
 7  OCBNAM EQU 9          WORD 9       OVERLAY NAME
 8  OCBSTK EQU 10                      BEGINNING OF STACK
 9  OCBSTL EQU 24                      LENGTH OF OCB STACKS
10  OCBLN  EQU 10+OCBSTL               LENGTH OF OCB
11  *
12  OCB:1A DFC $$         POINTER TO FIRST OCB
13  OCB:NU DFC $$         NUMBER OF OCB'S
14  OCB:CN DFC $$         AVAILABLE OCB'S (TWO'S COMPLEMENT)
15  OCB:WT DFC $$         POINTER TO TOP BCB AWAITING AN OCB
16  OCB:WL DFC $$         POINTER TO LAST BCB AWAITING AN OCB
17  OCB:CU DFC $$         CURRENT OCB
18  OCB:NE DFC $$         POTENTIAL OVERLAYS NEEDED
19  *
20  * EQUATES FOR OCBSTA
21  DP$    EQU B0         OVERLAY DOING PROCESSING
22  MA$    EQU B1         MEMORY ALLOCATED
23  PA$    EQU B2         PERMANENTLY ALLOCATED
24  OPA$   EQU B3         OPA READY FOR EXECUTION
```

BUFFER CONTROL BLOCK (BCB)

```
 1        EJT
 2  * BCB OFFSETS
 3  BCBSTA EQU 0          WORD 0       STATUS
 4  BCBBUF EQU 1          WORD 1       BUFFER ADDRESS
 5  BCBPNT EQU 2          WORD 2       OCB POINTER OR WAITING LINK
 6  BCBATT EQU 3          ATTENTIVE TASK NAME
 7  BCBUFT EQU 5          UFT
 8  BCBTIM EQU 13         START OF OPERATION TIMER DOUBLEWORD
 9  *
10  BCBLN  EQU 15         BCB LENGTH
11  *
12  BCB:1A DFC $$         POINTER TO FIRST BCB
13  BCB:NU DFC $$         NUMBER OF BCB'S
14  BCB:CN DFC $$         AVAILABLE BCB'S (TWO'S COMPLEMENT)
15  BCB:CU DFC $$         CURRENT BCB
16  BUFADR DFC $$         BEGINNING OF BUFFER AREA
17  BUFEND DFC $$         END OF BUFFER AREA
18  BCB:HE EQU OCB:WT
19  BCB:TA EQU OCB:WL
20  *
```

```
21  * EQUATES FOR BCBSTA
22  IU$     EQU B0        BCB IN USE
23  WO$     EQU B1        WAITING FOR OVERLAY
24  GU$     EQU B2        GIVE UP EXCLUSIVE USE
25  ST$     EQU B7        STATE = B4-7
26  STISO   EQU #F0FF     TO ISOLATE STATE FIELD
27  STEXT   EQU #0F00     TO CLEAR OUT STATE FIELD
28  STWP    EQU #0100     WRITING PROMPT
29  STRC    EQU #0200     READING COMMAND
30  STFO    EQU #0300     FINAL COMMAND
31  STIO    EQU #0400     INTERMEDIATE OUTPUT
32  STEM    EQU #0500     ERROR MESSAGE
33  REBCB   EQU #7F00     REINITIALIZE BCB STATUS FOR RESTART
34  *
35  BUFLN   EQU 36                READ BUFFER LENGTH (IN WORDS)
```

APPENDIX H
LOGGING NODE DATA

This appendix lists the data involved in each logging node. The data is entered by the user.

Unless otherwise noted, Word Ø of the node contains the negative number of minutes until midnight and Word 1 contains the negative number of ticks left in the current minute.

MEMORY PARITY ERROR - Format Number = 1, Length = 8 words

Word 2 Bits Ø-7 Parity Error Message ID (#Ø2)

Bits 8-15 Entry Condition Codes (NZOC)

Word 3 Most recent copy of Plane Status Register (PSR) Ø

Word 4 Erring data word read from memory PSR1

Word 5 Bits Ø-1Ø Set to zero

Bits 11-15 Extended Memory Module Address of the erring data from PSR2

Word 6 Memory address of the erring data from PSR3

Word 7 Bits Ø-5 Checks bits from PSR4

Bits 6-11 Error code from PSR4

Bits 12-15 Set to zero

BAD MEMORY PAGE DELETED - Format Number = 2, Length = 3 words

Word 2 Actual page number of bad memory page

TASK ABORT - Format Number = 34, Length = 12 words

Word 2 Task name MSH CAN code

Word 3 Task name LSH CAN code

Word 4 Overlay name MSH CAN code

Word 5 Overlay name LSH CAN code

Word 6 Abort address

Word 7 Program Status Word (PS) of aborted task

Word 8 Why task aborted

Word 9 Who aborted task

Word 10 CPU utilization timer MSH

Word 11 CPU utilization timer LSH

DEVICE OFF-LINE - Format Number = 5, Length = 6 words

Word 2 Device name CAN code

Word 3 Reason for being off-line CAN code

Word 4 Task name MSH CAN code (zero if not File Manager device)

Word 5 Task name LSH CAN code (zero if not File Manager device)

TASK ACTIVATION - Format Number = 6, Length = 7 words

Word 2 Task name MSH CAN code

Word 3 Task name LSH CAN code

Word 4 Global load module file name

Word 5 Execution priority

Word 6 Limit priority

TASK EXIT - Format Number = 7, Length = 10 words

Word 2 Task name MSH CAN code

Word 3 Task name LSH CAN code

Word 4 Overlay name MSH CAN code

Word 5 Overlay name LSH CAN code

Word 6 Why task exited CAN code

Word 7 Who made task exit CAN code

Word 8 CPU utilization timer MSH

Word 9 CPU utilization timer LSH

UNSOLICITED INTERRUPT - Format Number = 9, Length = 8 words

Word 2-6 Message data

Word 7 Fill

SUCCESSFUL I/O RETRY - Format Number = 10, Length = 12 words

Word 2 Positive count of retries made

Word 3 REX service number of operation

Word 4-6 DPI where failure occurred. (Sector/Track/Cylinder if disc)

Word 7 Transport name CAN code

Word 8 Status of controller at time of failure

Word 9 Device-dependent code

Word 10 Task name MSH CAN code

Word 11 Task name LSH CAN-Code

INTERTASK ERROR - Format Number = 11, Length = 11 words

Word 2 Error number

Word 3 Register 6

Word 4 Register 7

Word 5 Register 10

Word 6 Register 11

Word 7 Register 12

Word 8 Register 13

Word 9 Register 14

Word 10 Register 15

INTERTASK ERROR - Format Number = 12, Length = 4 words

Word 2 Error number

Word 3 ASCII string pertaining to error

FLUSH CURRENT LOGGING BUFFER REQUESTED BY REX - Format Number = 16, Length = 0 words

(This node type contains no node specific data.)

LOGGING RECORD STRUCTURE

HEADER

Word 0 Sequence number

Word 1 Length of specific data/format number

Word 2 Minutes until midnight

\ Word 3 Ticks left in minute

Word 4-N Up to 10 words of specific data

PHYSICAL RECORD

- Size of one sector
- Filled with logical records
- Terminated by #FFFF

FORMAT CHARACTERS

- %A Move ASCII string until null byte detected
- %C Convert CAN code to ASCII
- %D Convert one word to decimal ASCII
- %E Convert two words to decimal ASCII
- %H Convert to hexadecimal ASCII
- %T Convert two words of elapsed ticks to hours, minutes, seconds, ticks
- %U Convert two words of minutes and ticks until midnight to hours, minutes, seconds, ticks

MODCOMP SUPPLIED FILE: "EVEFOR" ON MC

Must be uncompressed prior to use.

FORMAT NUMBER	FORMAT DATA
0	RESERVED FOR MAX IV OPERATING SYSTEM INTERNAL USE: 0
1	%U PAR ID/CC=#%H PSRS: #%H #%H #%H #%H #%H
2	%U BAD MEMORY PAGE #%H DELETED
3	RESERVED FOR MAX IV OPERATING SYSTEM INTERNAL USE: 3
4	%U ABORT: %C%C/%C%C PSD=#%H/#%H WHY=%C WHOO=%C UTL=%T
5	%U OFFLINE: DEVICE=%C REASON=%C TASK=%C%C
6	%U ACTIVATE: %C%C FILE=%C PRI=%D LIM=%D
7	%U EXIT: %C%C/%C%C WHY=%C WHO=%C UTL=%T
8	RESERVED FOR MAX IV OPERATING SYSTEM INTERNAL USE: 8
9	%U UNSOLICITED INTERRUPT: %A
10	%U IO: C=#%H R=#%H A=#%H%H T=%C S=#%H%H %C%C
11	%U I: %D S=#%H %H %H %H %H %H %H %H %H
12	%U I: %D: %A
13	RESERVED FOR MAX IV OPERATING SYSTEM INTERNAL USE: 13
14	RESERVED FOR MAX IV OPERATING SYSTEM INTERNAL USE: 14
15	RESERVED FOR MAX IV OPERATING SYSTEM INTERNAL USE: 15
16	%U FLUSH OF CURRENT LOGGING BUFFER REQUESTED BY REX, EVELOG

APPENDIX I
ITC SYSTEM EQUATES

This appendix lists equates for use by the ITC user. The equates are located in IVSEQU. They cover the returned status codes, reserved close reason codes, and message types.

RETURNED STATUS CODES

```
ITEILB EQU          7         RECEIVER SHOULD IGNORE LAST BYTE
****** EQU          6         SERVER REPLIED CLOSE TO WAIT OPEN
ITESNC EQU          5         VC SLOTS NOT CONNECTED
ITESRD EQU          4         SERVER USER DATA REDEFINED
ITEPRD EQU          3         PORT REDEFINED
ITENMS EQU          2         NO MESSAGE FOUND
ITEMOR EQU          1         MORE DATA IN LOGICAL RECORD
ITEOK  EQU          Ø         NO ERROR
****** EQI          -1        RESERVED
ITEIBF EQU          -2        INVALID BUFFER ADDRESS OR LENGTH
ITEIVC EQU          -3        INVALID VIRTUAL CIRCUIT
ITEIDP EQU          -4        INVALID DESTINATION PORT
ITEMFU EQU          -5        MEMORY FULL (ITC LIMIT)
ITEMVC EQU          -6        ATTEMPT TO CHANGE MAXVCS
****** EQU          -7        RESERVED
ITECNS EQU          -8        CALLER HAS NO FREE SLOT
ITESNS EQU          -9        SERVER HAS NO FREE SLOT
ITEPVI EQU          -1Ø       PROTOCOL VIOLATION
****** EQU          -11       RESERVED
ITENMP EQU          -12       NO PORT FOUND
ITEIMV EQU          -13       INVALID MAXVCS FOR PORT SPECIFIED
****** EQU          -14       RESERVED
****** EQU          -15       RESERVED
ITEAAA EQU          -16       ARRIVAL ACTION ACTIVATE FAILURE
ITEIFN EQU          -17       INVALID ACTIVATION FILENAME
****** EQU          -18       RESERVED
****** EQU          -19       RESERVED
ITERNF EQU          -2Ø       SERVER NOT FOUND (VCOPEN)
****** EQU          -21       RESERVED
****** EQU          -22       RESERVED
****** EQU          -23       RESERVED
****** EQU          -24       RESERVED
ITEISG EQU          -25       INVALID SEGMENT IN LIST
ITEISL EQU          -26       INVALID SEGMENT LIST
****** EQU          -27       RESERVED
****** EQU          -28       RESERVED
ITEMTL EQU          -29       MESSAGE OVERFLOW FLUSHED
ITEART EQU          -3Ø       ATTEMPT TO REDEFINE SYSGEN SERVER DEF
****** EQU          -31       RESERVED
****** EQU          -32       RESERVED
****** EQU          -33       RESERVED
****** EQU          -34       RESERVED
ITESNF EQU          -35       SERVER NOT FOUND (SDINFO)
ITERUP EQU          -36       ATTEMPT TO REDEFINE USER OR SYSGEN PORT
ITEPBD EQU          -37       PORT BEING DEFINED ELSEWHERE
ITEIVP EQU          -38       INVALID REX PARAMETER
```

```
ITEICS EQU          -39         INVALID CHARACTER IN SERVER
ITEVSF EQU          -40         VC SLOT IS FREE
ITEWSZ EQU          -41         INVALID WINDOW SIZE
****** EQU          -42         RESERVED
ITEIWA EQU          -43         INSUFFICIENT WINDOW AVAILABLE
ITEWCL EQU          -44         FLOW WINDOW IS CLOSED
```

CLOSE REASON CODES

A reason code is supplied by a VCCLOSE. This code can be one of the system defined codes shown below and in the documentation corresponding to the appropriate system task, or it may be supplied by the user. Presently the definitions are:

```
              .           .          \
              .           .           \ UNASSIGNED
              .           .          /
             256                     /
             128 - 255     RESERVED FOR X25
              96 - 127     RESERVED FOR MAX IV NX
               1 -  95     RESERVED FOR FUTURE MAX IV
               0           UNASSIGNED
ITREXI EQU    -1           VCCLOSE REASON RETURNED ON EXITS
ITRABO EQU    -2           VCCLOSE REASON RETURNED ON ABORTS
ITRAAF EQU    -3           ACTIVATE FAILED ON PREVIOUS OPEN

              -4                      \
               .                       \ UNASSIGNED
               .                       /
               .                      /
```

MESSAGE TYPES

```
ITTDVC EQU          0           DATA ON VIRTUAL CIRCUIT
ITTMDA EQU          1           USER-SPECIFIED MORE DATA
ITTQDA EQU          2           USER-SPECIFIED QUALIFIED DATA
ITTMQA EQU          3           MORE QUALIFIED DATA
ITTOPE EQU          4           OPEN MESSAGE
ITTACC EQU          5           ACCEPT MESSAGE
ITTCLO EQU          6           CLOSE MESSAGE

ITTFWO EQU          12          FLOW CONTROL WINDOW OPEN
ITTVCR EQU          13          VC RESET OCCURED
```

APPENDIX J
REVISION H TO REVISION I CONVERSION NOTES

SYSGEN

PART 1 - SYSTEM STRUCTURE BLOCK

NEW - The TOCLOCK statement provides security which makes the online compression of load module files safe.

NOTE: TOCLOCK Ø saves MAP Ø space but removes security for online load module file manipulation.

The TASKMASTER SYSGEN has a new option ROLL-ON-HOLD. This has been added to change the ROUNDROBIN algorithm to benefit a multi-batch environment.

The SYSOPT statement has been modified to include the enabling of the date/time stamp option on messages and to include a level Ø power interrupt handler that consumes less MAP Ø space and that handles only DC power interrupts.

PART 2 - BASIC I/O SYSTEM BLOCK

Remove WORKBENCH macro from Part 2. This macro is no longer needed. The WORKBENCH macro in Part 1 is now sufficient.

PART 3 - TASK STRUCTURE BLOCK

The Exceptional Condition Task can have an additional LOGFILE assignment if the user has custom batch overlays. The logical file UBM should be assigned to the load module file where the custom overlays reside. Job Control will search UBM if present before searching the default load module file (BM).

FILE MANAGER

NEW - The $LFILE Directive was added to File Manager to list the contents of any File Manager directory, data or partition data file.

The $FDx Directive has been modified to list the contents of a user's File Descriptor List.

<u>All File Manager directives:</u>

- have an option (BS) to specify that the requested service is to be performed without using the default SYSGENed File Descriptor List (FDL).
- can contain an embedded volume name within a specified filename.
- can contain a filename (or volume name for volume related services) in place of a FDL specification (FDx). Other

descriptors can be specified after this filename. This allows a File Manager call without previously defining an FDL.

All File Manager errors:

- o list a text error message as well as the hexadecimal error code.

TOC

Revision L.Ø of the MAX IV Task/Overlay Cataloger (TOC) supports two new directives:

- o SQU - allows the user to move one module at a time in order to consolidate as much free space together as possible.
- o CLE - a command line extension to the TCB.

The CHA command has been removed. Use the CPD, JM, TMP or OC overlay.

Directory manipulating directives lock out the load module file during the manipulation of the directory when TOCLOCK is enabled. If TOCLOCK Ø is specified in Part 1 of the SYSGEN, the lockout feature is NOT enabled.

Users must do one of the following before cataloging any new programs with revision K.Ø or subsequent revisions of TOC.

- o Delete all segmented programs and then recatalog them with revisions subsequent to revision K.Ø of TOC.
- o GET all programs and then PUT all programs under either revision of TOC.
- o Compress the load module file under either revision of TOC.

JOB CONTROL

The following is a list of JM overlays that have been added in MAX IV Revision I.Ø.

AVA List available terminal ports.
BRO Broadcast a message to all terminals.
BOX Print a banner page before a listing
CPD Change Partition Definition
FAT Print a formatted list of file assignments
FOL File manager files offline.
FSP File manager volume space.
MSG Send a message to a terminal
TNA Print a list of system disc transports.
WHO Displays who is logged onto system.

Job Control now looks for a module on UBM (if present) before searching the default load module file. Refer to the $EXE Directive in the Job Control manual for more information.

Job Control now allows command line arguments to be passed to a processor or a JM overlay. This is facilitated by placing them after an ! on the command line.

INDEX

ABORT, 104
Abort messages, 273
Access Rights, 59, 68
Addressing, 57
ALO - LPR, 226
Alternate Track Processor, 20
ASSIGN, 105
AVA, 106
Available Memory Pool, 27
AVFILE, 107
AVRECORD, 108

Basic I/O System, 21
Basic Input/Output System Handlers, 21
Batch Processing, 265
Batch-processing tasks, 16
BAU, 109
Binary Record Dump, 20
BIOS, 21
BKFILE, 110
BKRECORD, 111
BRO, 112
Buffer Control Block, 285
Buffer Size, 23

Context Switching, 53
Controller Names, 249
CPR - LPR, 219
CTA - LPR, 232

DCA - LPR, 236
DDL - LPR, 208
DEA - LPR, 228
DEFAULT, 105
DEV, 113
Device Names, 249
DFL - LPR, 239
Direct Access Maintenance Processor, 20
Disc Layout, 49
DMR - LPR, 228
DPS - LPR, 235
DTV - LPR, 214
DUC - LPR, 237
Dump Analyzer, 20
DVE - LPR, 236

EGS - LPR, 218
EPS - LPR, 221
Equates, 291
ERA - LPR, 225
Error Messages, 154, 165, 271
ESM - LPR, 224
Established Tasks, 13

Event Logging, 145
Event Logging Report, 152
Exceptional Condition (X) Task, 29
Executive Services, 9

FAD - LPR, 238
FAR - LPR, 229
Fault Isolation, 283
FAV - LPR, 238
File Assignment Table (FAT), 22
File Manager, 25
File Names, 261, 265
FOL, 115

Global File Assignment Table (GFAT), 22
Global Logical File Names, 261

Hardware, 33
HOLD, 116
HOME, 117

ICM - LPR, 231
IGS - LPR, 217
Influence Levels, 14
Interrupts, 21, 89
IPS - LPR, 220
ISM - LPR, 222
ITC, 291

Job Control Procedures, 17
Job Streams, 18

KILL, 104

LEO, 204
LET, 123
Line Monitor Event, 101
Link Editor, 19
Link Loader, 20
Link-Editor, 26
Load Error, 279
load module definition, 201
Load Module Directory (LMD), 27
LOAD PROGRAM RECORDS, 203
Load Program Records (LPR's), 27
Loading, 26
Loading uncataloged programs, 19
Local Page, 65
Log Points, 150
Logging Nodes, 150, 287
Logical Device Table, 21
Logical Devices, 21
Logical File Names, 261, 265
logical Files, 23

Mapping, 58
MAR - LPR, 215

Memory Maps, 11
Memory requirements, 34
Memory, Access Rights, 28
Memory, allocation, 27
MES, 168
Messages, 148
MODULE LOADER ERROR MESSAGE CODES, 244
Module Loader Trace, 245
Module Object Records (MOR's), 27, 202
Modules, 76
Moving-Head Disc Preparation Program, 20
MSG, 118
MSGGET, 182
Multiprocessing, 2

Non-MAP 0 Resident Code, 45
nonresident programs, contiguous memory, 11
NOP - LPR, 230
NUM, 119

OCP, 120
OPTION, 121
OSCAT, 141
OSCST, 141
OSGDN, 142
OSPAR, 139
OSREL, 139
OSRER, 139
OSRET, 139
OSRWB, 139
OSRWL, 139
OSSOM, 142
OSTCB, 141
OSTID, 141
OSWRI, 140
Overlay Control Block, 285
Overlay Service Routines, 139
Overlays, 13, 135

Page Sharing, 70
PAS, 122
Peripheral Devices supported, 22
POPTION, 121
Power Failure Interrupt, 89
Pre-scheduled Tasks, 12
Priority Levels, 10, 14
Procedures, 17
PRT, 169
PRTDEF, 193
Push Stack, 70

RELEASE, 123
Resident Tasks, 12
RESUME, 123
REW, 124
REX Services, 72
RLP - LPR, 211

RMO-LPR, 210
ROC, 125
Roller Task, 28
Root Tasks, 11, 13
RUN, 126

SAC - LPR, 230
SDF, 167
SDI, 167
SDINFO, 191
Shared Page, 65
shared-memory, 2
SNV - LPR, 212
SOB - LPR, 232
Software Protection Trap, 93
Source Editor, 20
SPD - LPR, 233
SPL - LPR, 240
SPO - LPR, 212
Spooling, 18, 20
SSO - LPR, 211
STOP, 127
SVR, 169
SVRDEF, 195
Symbionts, 21
SYS, 128
SYSGEN, 147, 161
System Device Table, 25
System Generation (SYSGEN), definition of Phases, 4
System Logical Device Table, 23
System Tasks, 12

TAS, 129
Task, 2
Task Schedulers, 14
Task/Overlay Cataloger, 19
TASKMASTER, 55
TER, 130
Terminal Monitor Program, 97
Timers, 14
TMP, 97
Transient Non-Resident Tasks, 12

UDC - LPR, 240
Unimplemented Instruction Trap, 95
Unprivileged mode, 16
User Hook, 101
User Tasks, 11
USR - LPR, 242
UTI - LPR, 207
UTO - LPR, 208

VCCLOSE, 178
VCDISABLE, 186
VCENABLE, 185
VCGET, 180
VCINFO, 189

VCLINK, 187
VCOPEN, 171, 173
VCRESET, 197
VCSEND, 175

WAR - LPR, 214
WEOF, 131
WHO, 132
WKB - LPR, 242
WTH, 133

Fol

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 3624 FT. LAUDERDALE, FL 33309

POSTAGE WILL BE PAID BY ADDRESSEE

MODULAR COMPUTER SYSTEMS
1650 W. McNAB ROAD
P.O. BOX 6099
FT. LAUDERDALE, FLORIDA 33310

ttention: TECHNICAL PUBLICATIONS, M.S. #85

Fol

MODCOMP

Please comment on the publication's completeness, accuracy, and readability. We also appreciate any general suggestions you may have to improve this publication.

If you found errors in this publication, please specify the page number or include a copy of the page with your remarks.

Your comments will be promptly investigated and appropriate action will be taken. If you require a written answer, please check the box and include your address below.

☐

Comments: _____

Manual Title _____

Manual Order Number _____ Issue Date _____

Name _____ Position _____

Company _____

Address _____

_____ Telephone ( ) _____

MCS 148D (1/83)

MODCOMP

Corporate Headquarters:
MODULAR COMPUTER SYSTEMS, Inc., 1650 West McNab Road, P.O. Box 6099, Ft. Lauderdale, FL 33310, Tel: (305) 974-1380, TWX: 310-372-7837

International Headquarters:
MODULAR COMPUTER SERVICES, Inc., The Business Centre, Molly Millars Lane, Wokingham, Berkshire, RG11 2JQ, UK, Tel: 0734-786808, TLX: 851849149

Latin American Sales Headquarters:
MODULAR COMPUTER SYSTEMS, Inc., 1650 West McNab Road, P.O. Box 6099, Ft. Lauderdale, FL 33310, Tel: (305) 975-6562, TLX: 3727852

Canadian Headquarters:
MODCOMP Canada, Ltd., 400 Matheson Blvd. East, Unit 24, Mississauga, Ontario, Canada L4Z 1N8, Tel: (416) 890-0666, TELEX: 06-961279

SALES & SERVICE LOCATIONS THROUGHOUT THE WORLD

"The technical contents of this document, while accurate as of the date of publication, are subject to change without notice."

Printed in the U.S.A.